

Cocoa Supplement

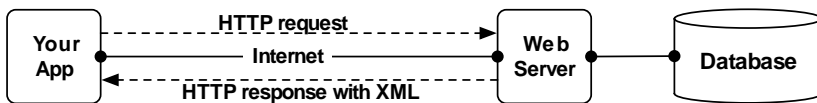
Cocoa Supplement

1. Web Services	1
DegreeDaze	1
Layout Interface	2
Data Model	4
Fetching the XML Data	6
Parsing the XML	7
2. Developing for the iPhone	11
Porting DegreeDaze to the iPhone	11
ClimateParser	13
RootViewController	14
UINavigationController	19
DegDazePhoneAppDelegate	19
UITableViewController	20
Pushing a View Controller onto the Navigation Stack	22
Core Location	22

Chapter 1. Web Services

Web services are getting a lot of hype. In the end, however, they are just an HTTP request and response where each may be carrying XML data. So using a web service from Cocoa is just a matter of being able to send HTTP requests and receive responses. It also may require generating and parsing XML.

Figure 1.1. Web Services



Generating and parsing XML can be done two ways:

- Low-level parsing is done using **NSXMLParser**. As it parses an **NSData** containing XML, it makes calls to its delegate as it encounters different XML structures.
- High-level parsing is done with **NSXMLDocument** and **NSXMLNode**. In high-level parsing, the entire **NSData** is parsed into a tree of **NSXMLNode** objects.

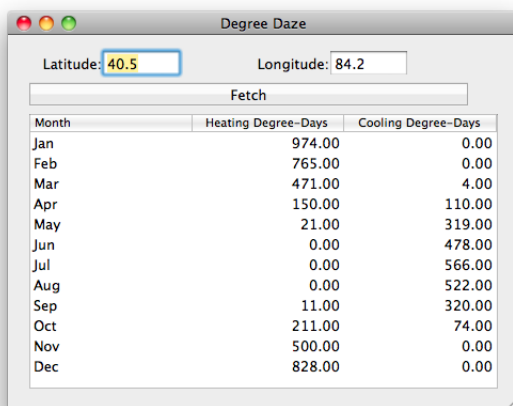
You will be using low-level parsing. It is a little more complicated to use, but **NSXMLNode** and **NSXMLDocument** don't exist on the phone. Thus, to learn the more portable method, you will be using **NSXMLParser** directly.

DegreeDaze

In this exercise, you are going to write an application that uses the data from the United Nations Environment Programme. For a given latitude and longitude, the UNEP can tell you the climate at that location. For each month, you can get the degree-days below 18 degrees Celsius—this gives you a good idea of how much you will need to spend to heat a building in that location. You can also get the degree-days above 10 degrees Celsius—this gives you a good idea of how much you will spend to cool a building at that location.

Here is what the application will look like:

Figure 1.2. DegreeDaze



Create a new project of type Cocoa Application. Name it DegreeDaze.

In DegreeDazeAppDelegate.h declare five outlets and an action:

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

@interface DegreeDazeAppDelegate <NSApplicationDelegate, NSTableViewDataSource>
{
    NSWindow *window;
    IBOutlet NSTextField *latField;
    IBOutlet NSTextField *lonField;
    IBOutlet NSTableView *tableView;
    IBOutlet NSButton *fetchButton;
    IBOutlet NSProgressIndicator *progressIndicator;
}

@property (assign) IBOutlet NSWindow *window;

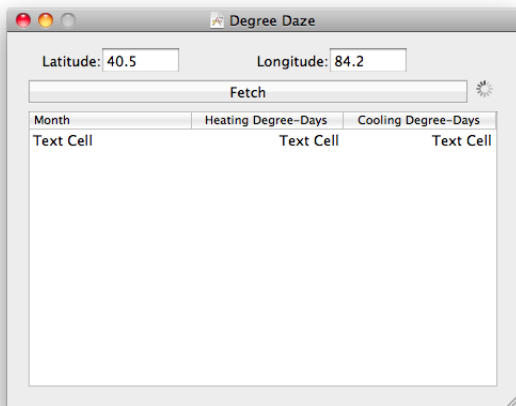
- (IBAction)fetchCancel:(id)sender;

@end
```

Layout Interface

Open MainMenu.xib. Drop a few text fields, a button, a progress indicator, and a table view on the window. Give the table view three columns:

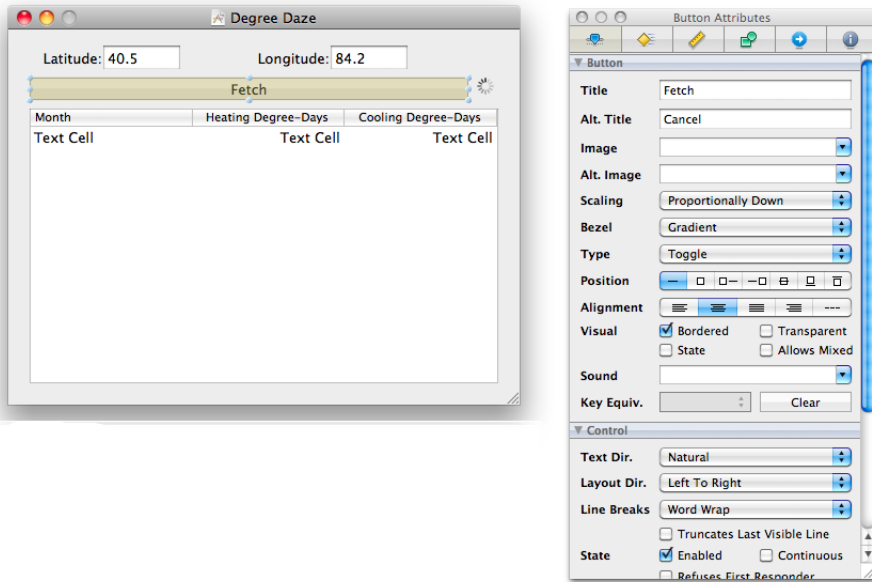
Figure 1.3. Layout



Set the identifiers on the table view columns to month, clim_cdd10_nasa_low, and clim_hdd18_nasa_low.

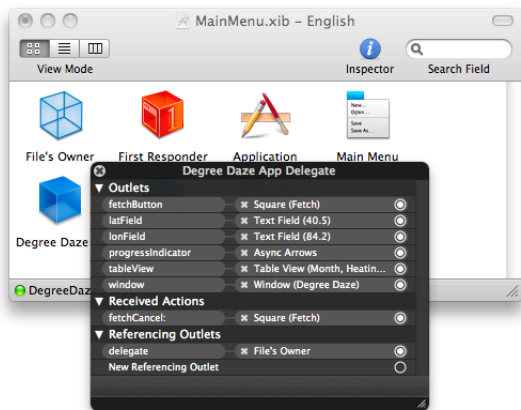
Control-drag to set the target and action of the button. It should trigger the **fetchCancel:** method of the **DegreeDazeAppDelegate** object. Also, the button type should be set to Toggle. The title should be Fetch and the Alt Title should be Cancel.

Figure 1.4. Button Inspector



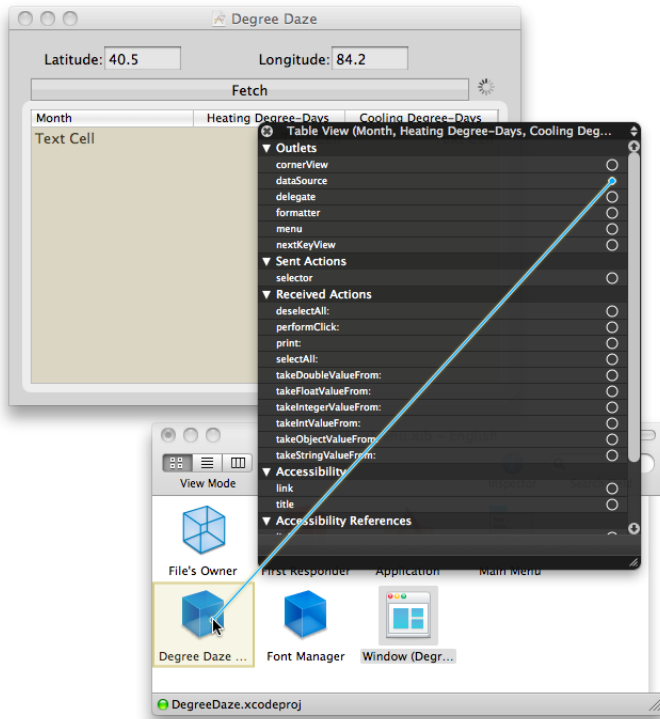
Control-click on the **DegreeDazeAppDelegate** to bring up the connection panel and connect all five outlets: tableView, latField, lonField, fetchButton, and progressIndicator.

Figure 1.5. Connections



Control-click on the table view to bring up the connection panel. Make **DegreeDazeAppInspector** the data source for the table view. (Don't see the dataSource outlet? Did you select the scroll view instead of the table view?)

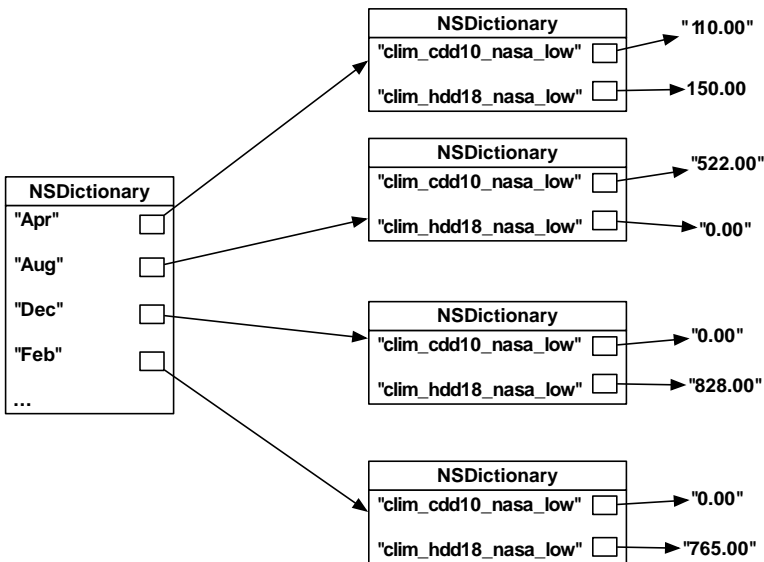
Figure 1.6. Data Source



Data Model

The data will be kept in a dictionary. The keys will be the names of months; the objects will be dictionaries. Those month dictionaries will have two entries: one for the cooling degree-days and another for the heating degree-days.

Figure 1.7. Data Model



Add an instance variable for the dictionary of months to DegreeDazeAppDelegate.h:

```
NSMutableDictionary *climateRecords;
```

A dictionary does not have an order, so you will need an array of month names in the expected order. Add a static variable near the top of DegreeDazeAppDelegate.m. Initialize both the array and the dictionary in the `init` method:

```
#import "DegreeDazeAppDelegate.h"

static NSArray *months;

@implementation HeatingCostAppDelegate

@synthesize window;

- (id)init
{
    [super init];

    // Does the months array need to be initialized?
    if (!months) {
        months = [[NSArray alloc] initWithObjects:@"Jan", @"Feb",
            @"Mar", @"Apr", @"May", @"Jun", @"Jul", @"Aug",
            @"Sep", @"Oct", @"Nov", @"Dec", nil];
    }

    // Fill the climateRecords dictionary with empty mutable dictionaries
    climateRecords = [[NSMutableDictionary alloc] init];
    for (NSString *monthName in months) {
        NSMutableDictionary *newRecord = [[NSMutableDictionary alloc] init];
        [climateRecords setObject:newRecord forKey:monthName];
        [newRecord release];
    }
    return self;
}
```

Implement the table view data source methods:

```
#pragma mark Table view data source methods

- (NSInteger)numberOfRowsInTableView:(NSTableView *)tv
{
    return [months count];
}

- (id)tableView:(NSTableView *)tv
objectValueForTableColumn:(NSTableColumn *)tc
row:(NSInteger)row
{
    NSString *identifier = [tc identifier];
    NSString *month = [months objectAtIndex:row];

    // Is this the month name column?
    if ([identifier isEqual:@"month"]) {
        return month;
    }

    // Get the dictionary for the appropriate month
    NSDictionary *cr = [climateRecords objectForKey:month];

    return [cr objectForKey:identifier];
}
```

When the window is closed, you want the application to terminate. Implement the appropriate delegate method:

```
#pragma mark Application delegate methods
```

```
- (BOOL)applicationShouldTerminateAfterLastWindowClosed:(NSApplication *)sender
{
    return YES;
}
```

Create a stub for the **fetchCancel:** method:

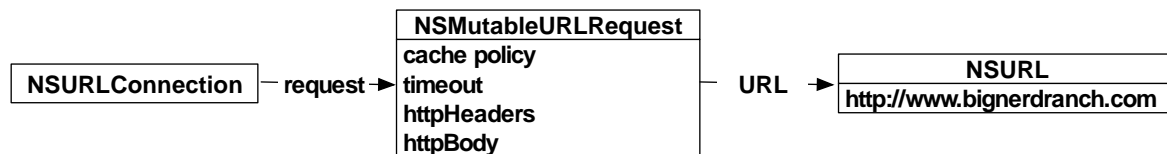
```
- (IBAction)fetchCancel:(id)sender
{
    NSLog(@"fetchCancel:");
}
```

Build and run the application. It doesn't do much yet, but it should list the months in the first column.

Fetching the XML Data

An **NSURLConnection** is used to fetch data via HTTP. It takes an **NSMutableURLRequest** object and a delegate. The delegate gets callbacks as the data arrives. An **NSMutableURLRequest** has an **NSURL**, a cache-usage policy, and a timeout.

Figure 1.8. URL Fetching



You will need a variable to hold on to the **NSURLConnection**. The data will come in gradually, so you will also need an **NSMutableData** that can be appended to. Declare the necessary instance variables in `DegreeDazeAppDelegate.h`:

```
// For fetch
NSMutableData *data;
NSURLConnection *connection;
```

There are a few places where you will need to clean up the connection, the **NSMutableData**, and the user interface during or after a fetch. Create a method near the top of `DegreeDazeAppDelegate.m`:

```
- (void)cleanUpFetch
{
    [connection release];
    connection = nil;
    [data release];
    data = nil;
    [fetchButton setState:NO];
    [progressIndicator stopAnimation:nil];
}
```

In **fetchCancel:** you will either fetch or cancel the fetch depending on whether a connection exists:

```
- (IBAction)fetchCancel:(id)sender
{
    // Is this a cancel?
    if (connection) {
        NSLog(@"canceling fetch");
    }
}
```

```

        [connection cancel];
        [self cleanUpFetch];
    } else {
        NSString *urlString = [NSString stringWithFormat:@"http://na.unep.net/swera_ims/WS/index.php?"
            "coords=%.3f,%.3f&layers=clim_hdd18_nasa_low,clim_cdd10_nasa_low",
            [latField floatValue], [lonField floatValue]];
        NSLog(@"fetching %@", urlString);

        NSURL *url = [NSURL URLWithString:urlString];

        NSURLRequest *request = [NSURLRequest requestWithURL:url
            cachePolicy:NSURLRequestReturnCacheDataElseLoad
            timeoutInterval:30];

        data = [[NSMutableData alloc] init];

        connection = [[NSURLConnection connectionWithRequest:request delegate:self] retain];
        [progressIndicator startAnimation:nil];
    }
}
}

```

Create delegate methods for the `NSURLConnection`:

```

- (void)updateInterfaceWithData:(NSData *)d
{
    NSString *xmlString = [[NSString alloc] initWithData:d encoding:NSUTF8StringEncoding];
    NSLog(@"Will parse: %@", xmlString);
    [xmlString release];
}

#pragma mark URLConnection delegate methods

- (void)connection:(NSURLConnection *)conn
    didReceiveData:(NSData *)d
{
    [data appendData:d];
}

- (void)connectionDidFinishLoading:(NSURLConnection *)conn
{
    [self updateInterfaceWithData:data];
    [self cleanUpFetch];
}

- (void)connection:(NSURLConnection *)conn didFailWithError:(NSError *)error
{
    UIAlertView *alert = [UIAlertView alertWithError:error];
    [alert runModal];
    [self cleanUpFetch];
}

```

Build and run the application. The app still doesn't do much, but you should see the fetched XML on the console when you fetch. Look for errors before going any further.

Parsing the XML

Now you are ready to parse the XML. Create a new class called `ClimateParser`.

In `ClimateParser.m`, create a method that creates an `NSXMLParser` and kicks off the parsing of the data:

```

- (BOOL)parseData:(NSData *)data
    intoDictionary:(NSDictionary *)dict
    error:(NSError **)errptr

```

```

{
    // Put the dictionary into a instance variable
    monthDict = dict;

    // Create a parser with the XML data
    NSXMLParser *parser = [[NSXMLParser alloc] initWithData:data];

    // Self will receive the callbacks
    [parser setDelegate:self];

    // Kickoff the parse
    BOOL successful = [parser parse];

    // Do I need to deal with an error?
    if (!successful && errptr != NULL) {
        *errptr = [parser parserError];
    }

    // Clean up
    [parser release];
    return successful;
}

```

As the parser parses through the data, it will make calls to the delegate methods:

```

#pragma mark XML Parser delegate methods

- (void)parser:(NSXMLParser *)p
  didStartElement:(NSString *)elementName
  namespaceURI:(NSString *)namespaceURI
  qualifiedName:(NSString *)qName
  attributes:(NSDictionary *)attributeDict
{
    // Is this the start of a layer?
    if ([elementName isEqual:@"Layer"]) {
        currentLayer = [[attributeDict objectForKey:@"name"] copy];
        return;
    }

    // Is this the start of a month?
    if ([monthDict objectForKey:elementName]) {
        currentMonth = [elementName copy];
        currentString = [[NSMutableString alloc] init];
        return;
    }
}

- (void)parser:(NSXMLParser *)p
  didEndElement:(NSString *)elementName
  namespaceURI:(NSString *)namespaceURI
  qualifiedName:(NSString *)qName
{
    // Is this the end of a layer?
    if ([elementName isEqual:@"Layer"]) {

        // Clear currentLayer
        [currentLayer release];
        currentLayer = nil;
        return;
    }

    // Is this the end of a month?
    NSMutableDictionary *cr = [monthDict objectForKey:elementName];
    if (cr) {

        // The month is represented by a dictionary

```

```

        [cr setObject:currentString forKey:currentLayer];

        // Clear the currentString
        [currentString release];
        currentString = nil;

        // Clear the currentMonth
        [currentMonth release];
        currentMonth = nil;
        return;
    }
}

// This might get called several times within a single element
- (void)parser:(NSXMLParser *)p foundCharacters:(NSString *)string
{
    [currentString appendString:string];
}

```

In ClimateParser.h, declare the necessary instance variables and the one public method:

```

#import <Foundation/Foundation.h>

@interface ClimateParser : NSObject <NSXMLParserDelegate>
{
    NSDictionary *monthDict;
    NSString *currentLayer;
    NSString *currentMonth;
    NSMutableString *currentString;
}

- (BOOL)parseData:(NSData *)data
  intoDictionary:(NSDictionary *)dict
    error:(NSError **)errptr;

@end

```

Back in DegreeDazeAppDelegate.m, use the parser:

```

- (void)updateInterfaceWithData:(NSData *)d
{
    /*
    NSString *xmlString = [[NSString alloc] initWithData:d encoding:NSUTF8StringEncoding];
    NSLog(@"Will parse: %@", xmlString);
    */

    ClimateParser *parser = [[ClimateParser alloc] init];
    NSError *error;

    if (![parser parseData:d intoDictionary:climateRecords error:&error]) {
        NSAlert *alert = [NSAlert alertWithError:error];
        [alert runModal];
    }

    NSLog(@"climateRecords = %@", climateRecords);

    [parser release];
    [tableView reloadData];
}

```

Import the header file at the top of DegreeDazeAppDelegate.m:

```
#import "ClimateParser.h"
```

Build and run the application.

Chapter 2. Developing for the iPhone

Applications for the iPhone are written using Xcode and *the Cocoa Touch Frameworks*. Cocoa Touch comprises Foundation, Core Graphics, and UIKit. UIKit is analogous to AppKit -- it supplies the windows, events, views, buttons, etc for iPhone programmers. UIKit is, however, not the same as AppKit. This chapter is designed to get you started with developing applications on the iPhone with an emphasis on what *is not the same*.

In particular, you will not have the garbage collector and will use the retain/release mechanism for memory management. You will use OpenGL ES instead of regular OpenGL. When one app starts up, the other usually terminates. Windows and table view cells are subclasses of **UIView**.

Porting DegreeDaze to the iPhone

Most of the stuff that makes DegreeDaze work (table views, **NSXMLParser**, **NSURLConnection**) exist on the iPhone. Porting it from Cocoa to Cocoa Touch will give you a feel for many of the differences between the two platforms. You will use three common iPhone OS features in your port: a navigation controller, a table view, and the Core Location framework. There will be two **UIViewController**s:

Figure 2.1. RootViewController

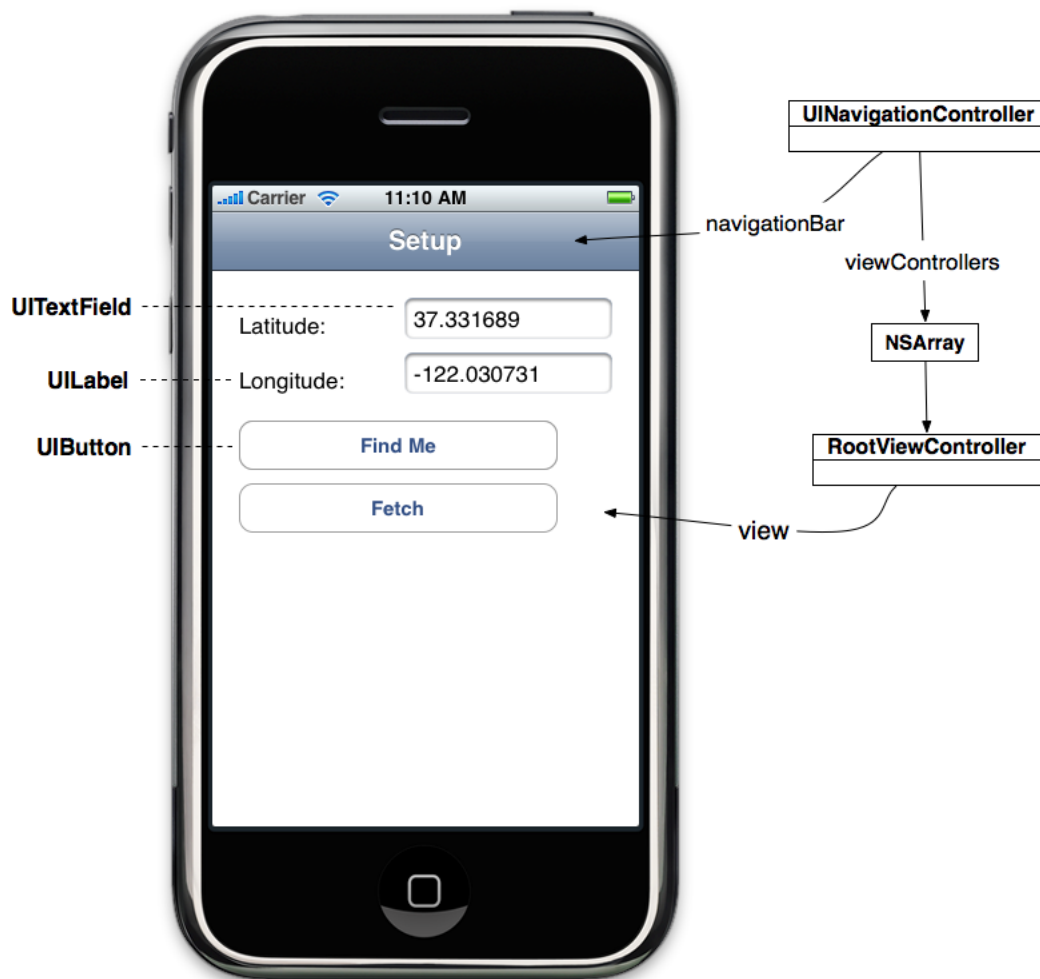
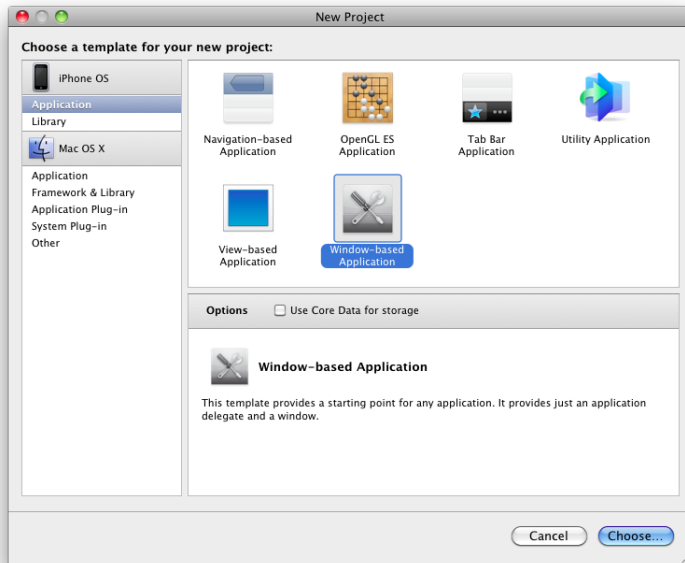


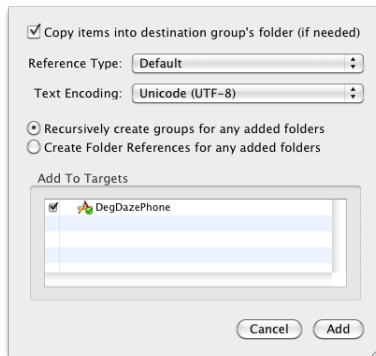
Figure 2.3. New iPhone OS Project



ClimateParser

You are going to use the same parser object you wrote earlier for the Mac version. Drag `ClimateParser.h` and `ClimateParser.m` from the DegreeDaze project into this project. Be sure to check the "Copy" checkbox.

Figure 2.4. Adding ClimateParser to project



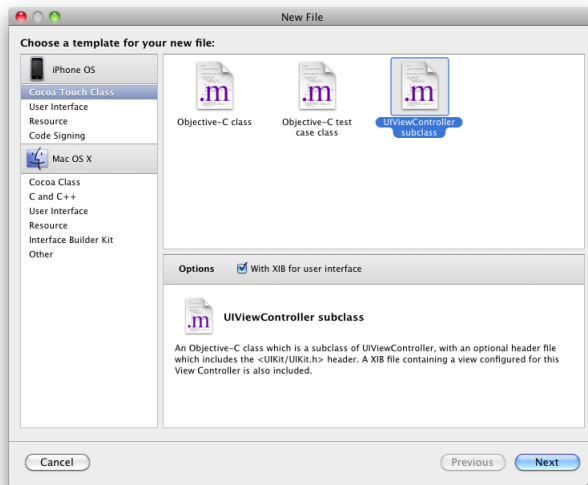
`NSXMLParser`'s delegate methods are declared in a category instead of a protocol on iPhone OS so remove the protocol declaration from `ClimateParser.h`.

```
@interface ClimateParser : NSObject {
    NSDictionary *monthDict;
    NSString *currentLayer;
    NSString *currentMonth;
    NSMutableString *currentString;
}
```

RootViewController

Create a new **UIViewController** subclass called **RootViewController**. Check the option box labeled "With XIB for user interface" so you can use Interface Builder to layout the subviews.

Figure 2.5. Subclassing UIViewController



Open `RootViewController.h`, edit it as follows, and save the file. Notice that `UIKit` has many parallel objects to `AppKit`, but in some cases the similarity is only as deep as the name. Be sure to check the iPhone OS documentation to discover the differences.

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

@interface RootViewController : UIViewController {
    IBOutlet UITextField *latField;
    IBOutlet UITextField *lonField;
    IBOutlet UIButton *findMeButton;
    IBOutlet UIButton *fetchButton;
    IBOutlet UIActivityIndicatorView *progressIndicator;

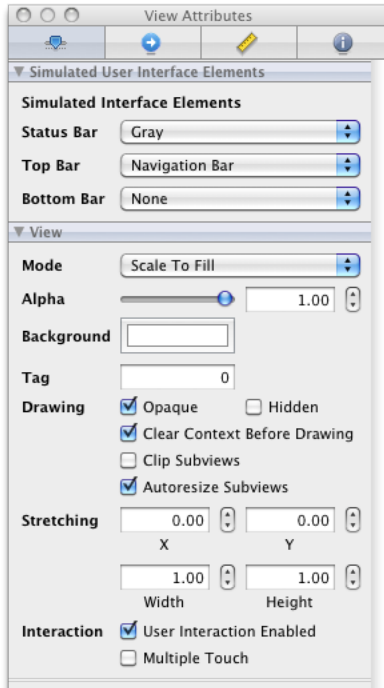
    NSMutableDictionary *climateRecords;

    // For fetch
    NSMutableData *data;
    NSURLConnection *connection;
}
- (IBAction)findMeCancel:(id)sender;
- (IBAction)fetchCancel:(id)sender;

@end
```

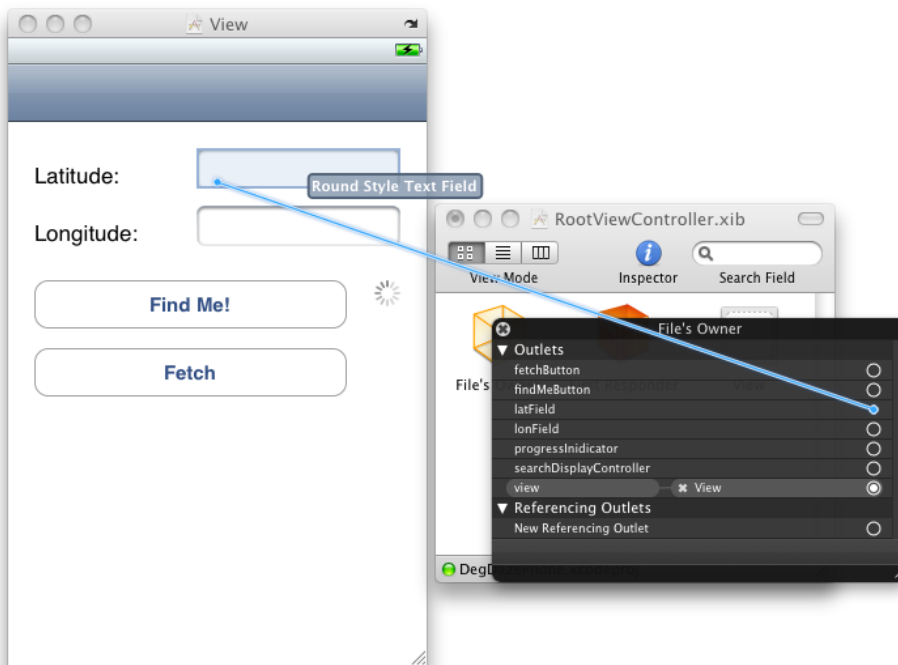
Open `RootViewController.xib`. Most iPhone OS applications contain similar UI elements provided by Apple such as a navigation bar or a tab bar. Interface Builder provides simulated interface elements for the **UIView** class that help you correctly layout subviews even if these elements are created elsewhere in the application. Open the View object in the inspector. Our application will use a navigation bar so set the "Top Bar" element accordingly.

Figure 2.6. Simulated Interface Elements



Layout the subviews as shown and make the connections with the File's Owner, which is the **RootViewController** class. Control-drag from the **UIButton**s to apply a touch to an action.

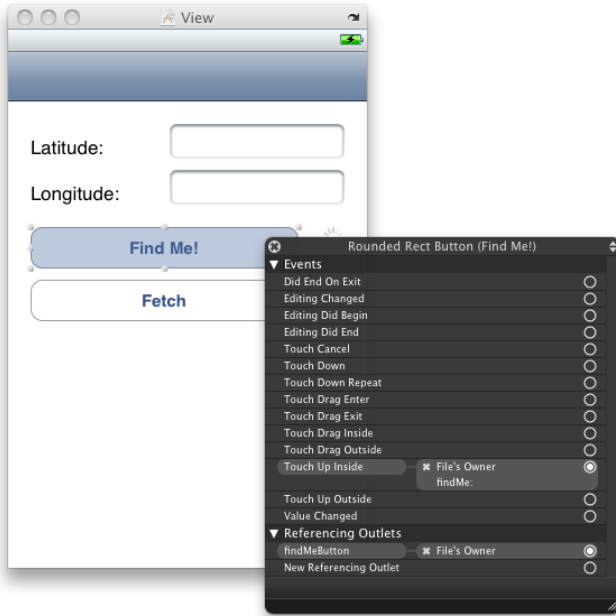
Figure 2.7. RootViewController.xib



Right-click a **UIButton** to inspect the list of control events already provided by Apple. The default control type is when a user lifts their finger inside the object's frame, but depending on the purpose of the object a different

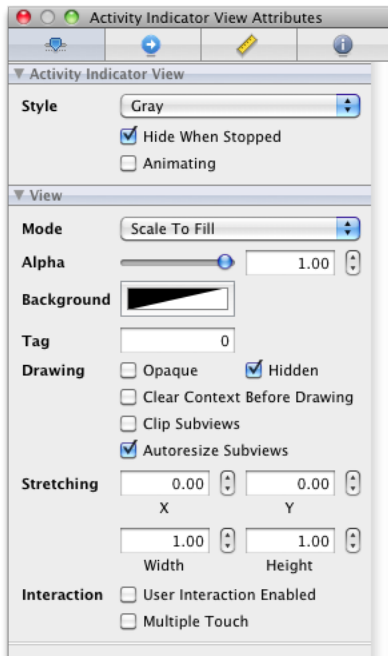
behavior might be desired. The list is thorough, but if you need a custom touch behavior not listed check the documentation for **UIResponder** (from which **UIButton** inherits from) to learn how to respond to touch events directly.

Figure 2.8. Control Events



Open the **UIActivityIndicatorViewView** in the inspector and select "Hide When Stopped".

Figure 2.9. UIActivityIndicatorView attributes



Save and close `RootViewController.xib`. Open `RootViewController.m`. You will see that Xcode has already generated the most commonly overridden methods of **UIViewController** for you. Add a static variable for the

months array near the top of the file, and override the designated initializer just as you did with the Mac version of DegreeDaze.

```
static NSArray *months;

@implementation RootViewController

- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil
{
    if (self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil] {
        // The navigation item holds information that the navigation
        // controller uses to update the appearance of the navigation bar
        [[self navigationItem] setTitle:@"Setup"];

        // Does the months array need to be initialized?
        if (!months) {
            months = [[NSArray alloc] initWithObjects:@"Jan", @"Feb",
                @"Mar", @"Apr", @"May", @"Jun", @"Jul", @"Aug",
                @"Sep", @"Oct", @"Nov", @"Dec", nil];
        }

        // Fill a climate records dictionary with empty dictionaries
        climateRecords = [[NSMutableDictionary alloc] init];
        for (NSString *monthName in months) {
            NSMutableDictionary *newRecord = [[NSMutableDictionary alloc] init];
            [climateRecords setObject:newRecord forKey:monthName];
            [newRecord release];
        }
    }
    return self;
}
}
```

For now just put a stub for the **findMeCancel:** method. Implement the **fetchCancel:** method, and the **NSURLConnection** delegate methods as you did on the Mac version. You can copy/paste most of the code from the other project, but there are a few changes in the iPhone OS version highlighted below. **NSAlert** objects have been replaced by **UIAlertView** objects, and **UIButton** has different methods than **NSButton** (touches and mouse clicks behave differently).

```
#pragma mark Find Me

- (IBAction)findMeCancel:(id)sender
{
}

#pragma mark Fetch

- (void)cleanUpFetch
{
    [connection release];
    connection = nil;
    [data release];
    data = nil;
    [fetchButton setTitle:@"Fetch" forState:UIControlStateNormal];
    [progressIndicator stopAnimating];
}

- (IBAction)fetchCancel:(id)sender
{
    if (connection) {
        NSLog(@"canceling fetch");
        [connection cancel];
        [self cleanUpFetch];
    } else {
}
```

```

NSString *urlString = [NSString stringWithFormat:
    @"http://na.unep.net/swera_ims/WS/index.php?"
    "coords=%.3f,%.3f&layers=clim_hdd18_nasa_low,clim_cdd10_nasa_low",
    [[latField text] floatValue], [[lonField text] floatValue]];
NSLog(@"fetching %@", urlString);

NSURL *url = [NSURL URLWithString:urlString];
NSURLRequest *request = [NSURLRequest requestWithURL:url
    cachePolicy:NSURLRequestReturnCacheDataElseLoad
    timeoutInterval:30];

data = [[NSMutableData alloc] init];

connection = [[NSURLConnection connectionWithRequest:request delegate:self] retain];

[fetchButton setTitle:@"Cancel" forState:UIControlStateNormal];
[progressIndicator startAnimating];
}
}

#pragma mark URLConnection delegate methods

- (void)updateInterfaceWithData:(NSData *)d
{
    ClimateParser *parser = [[ClimateParser alloc] init];

    NSError *error;
    if (![parser parseData:d intoDictionary:climateRecords error:&error]) {
        UIAlertView *alert = [[[UIAlertView alloc] initWithTitle:@"ClimateParser Error"
            message:[error localizedFailureReason]
            delegate:nil
            cancelButtonTitle:@"Dismiss"
            otherButtonTitles:nil] autorelease];

        [alert show];
    }

    NSLog(@"climateRecords = %@", climateRecords);

    [parser release];
}

- (void)connection:(NSURLConnection *)conn didReceiveData:(NSData *)d
{
    [data appendData:d];
}

- (void)connectionDidFinishLoading:(NSURLConnection *)conn
{
    [self updateInterfaceWithData:data];
    [self cleanUpFetch];
}

- (void)connection:(NSURLConnection *)conn didFailWithError:(NSError *)error
{
    UIAlertView *alert = [[[UIAlertView alloc] initWithTitle:@"NSURLConnection Error"
        message:[error localizedFailureReason]
        delegate:nil
        cancelButtonTitle:@"Dismiss"
        otherButtonTitles:nil] autorelease];

    [alert show];
    [self cleanUpFetch];
}
}

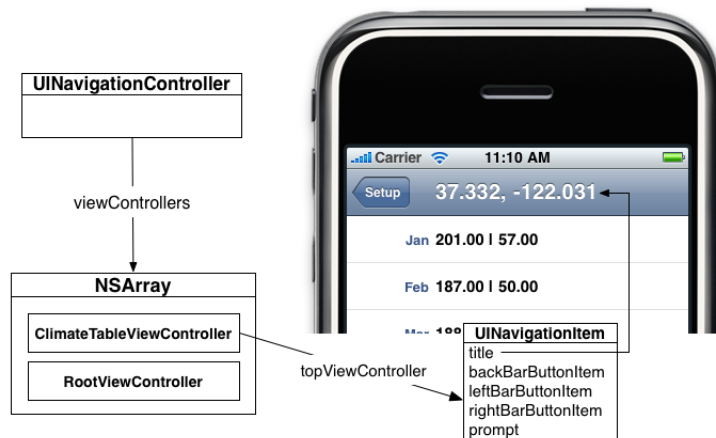
```

UINavigationController

Due to the iPhone's limited screen size your applications are limited to a single window. To get around this limitation Apple has provided two handy subclasses of **UIViewController** to manage multiple views elegantly: **UITabBarController** and **UINavigationController**. Nearly every iPhone OS application has either one or both of these controllers at work. **UITabBarController** is used to switch contexts between the various features of an application, and **UINavigationController** is used to "drill down" through the details of one particular feature. Together they provide a user experience much like switching windows on the desktop. You will use **UINavigationController** in your application.

UINavigationController manages a stack of **UIViewController** (or subclass) objects. It is initialized with a root view controller that resides at the bottom of the stack. When a new view controller needs to be presented to the user it is pushed onto the navigation controller's stack. **UINavigationController** automatically manages the navigation bar and updates its contents with the navigation item of the view controller at the top of its stack.

Figure 2.10. UINavigationController



Here is a quick list of the **UIViewController** methods you will use the most:

- (id)initWithRootViewController:(UIViewController *)rootViewController
- (void)pushViewController:(UIViewController *)viewController animated:(BOOL)animated
- (UIViewController *)popViewControllerAnimated:(BOOL)animated

DegDazePhoneAppDelegate

UIApplication takes the place of **NSApplication** on iPhone OS. It provides the launching point for all applications. Open DegDazePhoneAppDelegate.h and add an instance variable for a **UINavigationController**.

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

@interface DegDazePhoneAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    UINavigationController *navController;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;
@end
```

Open DegDazePhoneAppDelegate.m. When the application finishes launching we need to create a **RootViewController** instance and add it to the navigation controller's stack as the root view controller. Finally, we add the navigation controller's view to the window to be presented to the user.

```
#import "DegDazePhoneAppDelegate.h"
#import "RootViewController.h"

@implementation DegDazePhoneAppDelegate
@synthesize window;

- (void)applicationDidFinishLaunching:(UIApplication *)application {
    RootViewController *rootViewController = [[RootViewController alloc] initWithNibName:nil bundle:nil];

    UINavigationController *navController = [[UINavigationController alloc] initWithRootViewController:rootViewController];
    [rootViewController release];

    [window addSubview:[navController view]];
    [window makeKeyAndVisible];
}
```

Build and run the application in the simulator. You should be able to enter a latitude and longitude in the text fields, and when you tap fetch you should see the resulting climate record dictionary posted to the console. Now let's setup a table view to present the data to the user.

UITableViewController

Table views are found in nearly every iPhone OS application from the iPod to Stocks. They provide a highly customizable way of displaying ordered sets of information to the user. After working with table views for a bit you will quickly see that most of the attractive user interfaces in your favorite apps are simply table views.

UITableViewController is a subclass of **UIViewController** specifically designed to manage a **UITableView**. Create a new **UITableViewController** subclass called **ClimateTableViewController**. Open **ClimateTableViewController.h** and add instance variables to hold pointers to the information the table view will display. Declare the pointers as assigned properties.

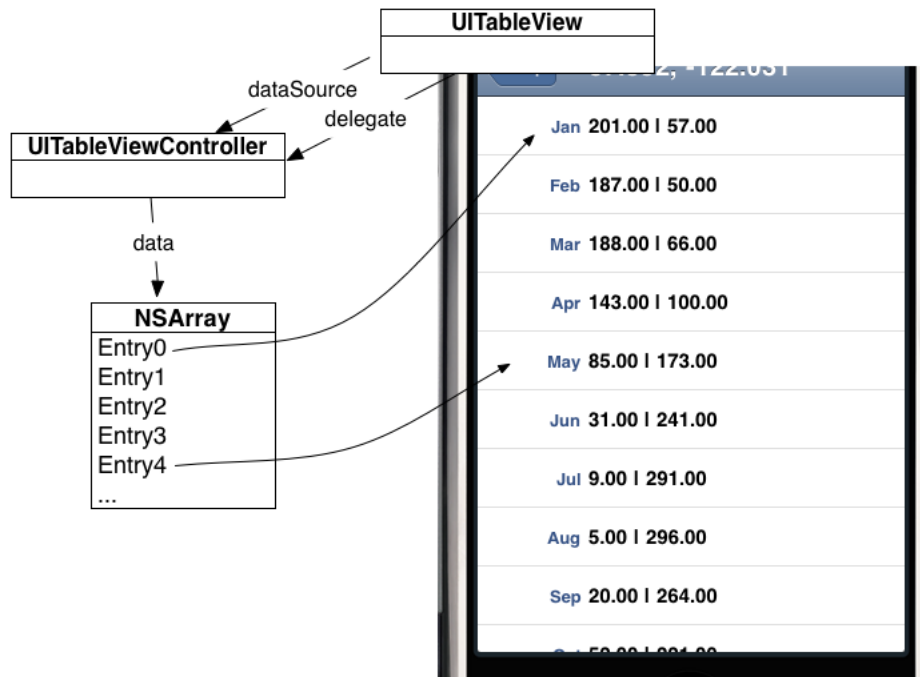
```
#import <UIKit/UIKit.h>

@interface ClimateTableViewController : UITableViewController {
    NSDictionary *data;
    NSArray *months;
}
@property (nonatomic, assign) NSDictionary *data;
@property (nonatomic, assign) NSArray *months;

@end
```

UITableViewController automatically creates the table view it manages. It is also the data source and delegate for the table view so your **UITableViewController** subclass must override the appropriate methods to get data on the table view. Consistent with data source and delegation methods in other Apple frameworks you do not give the table view data whenever you want. When the table view is ready to display data it will ask you for the necessary information.

Figure 2.11. UINavigationController



Open `ClimateTableViewController.m` and edit it as follows:

```
#import "ClimateTableViewController.h"

@implementation ClimateTableViewController

@synthesize data, months;

#pragma mark Table view methods

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return 1;
}

- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section {
    return [months count];
}

- (UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    static NSString *CellIdentifier = @"Cell";

    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier];
    if (cell == nil) {
        cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleValue2
            reuseIdentifier:CellIdentifier] autorelease];
        [cell setSelectionStyle:UITableViewCellStyleNone];
    }

    NSString *month = [months objectAtIndex:indexPath.row];
    NSDictionary *cr = [data objectForKey:month];

    NSString *hdd = [cr objectForKey:@"clim_hdd18_nasa_low"];
    NSString *cdd = [cr objectForKey:@"clim_cdd10_nasa_low"];
    NSString *detailText = [NSString stringWithFormat:@"%s | %s", hdd, cdd];
}
```

```
[[cell.textLabel] setText:month];
[[cell.detailTextLabel] setText:detailText];

return cell;
}
```

Pushing a View Controller onto the Navigation Stack

After the **NSURLConnection** completes its fetch we want to push the table view controller onto the navigation stack to display the climate records. Open `RootViewController.m` and edit the `updateInterfaceWithData:` method.

```
- (void)updateInterfaceWithData:(NSData *)d
{
    ClimateParser *parser = [[ClimateParser alloc] init];

    NSError *error;
    if (![parser parseData:d intoDictionary:climateRecords error:&error]) {
        UIAlertView *alert = [[[UIAlertView alloc] initWithTitle:@"ClimateParser Error"
                                                            message:[error localizedFailureReason]
                                                            delegate:nil
                                                            cancelButtonTitle:@"Dismiss"
                                                            otherButtonTitles:nil] autorelease];

        [alert show];
    }

    ClimateTableViewController *climateTable =
        [[ClimateTableViewController alloc] initWithStyle:UITableViewStylePlain];

    // Set the title of the table view to be location of the data
    NSString *latAndLong = [NSString stringWithFormat:@"%0.3f, %0.3f",
                            [[latField text] floatValue], [[lonField text] floatValue]];
    [[climateTable navigationItem] setTitle:latAndLong];

    // Set the data pointers for the table view
    [climateTable setData:climateRecords];
    [climateTable setMonths:months];

    // Push the table view controller on the navigation and stack and release it,
    // the navigation controller retains the view controller's on its stack
    [[self navigationController] pushViewController:climateTable animated:YES];
    [climateTable release];

    [parser release];
}
```

Build and run the application. After the fetch completes the table view should be pushed onto the stack.

Core Location

When designing iPhone OS applications it is best to minimize the amount of typing the user must do since many people find the cramped keyboard hard to deal with. It would be cool if the application found the current location of the user automatically so the user did not have to type in their latitude and longitude manually. Core Location is a service that does just that.

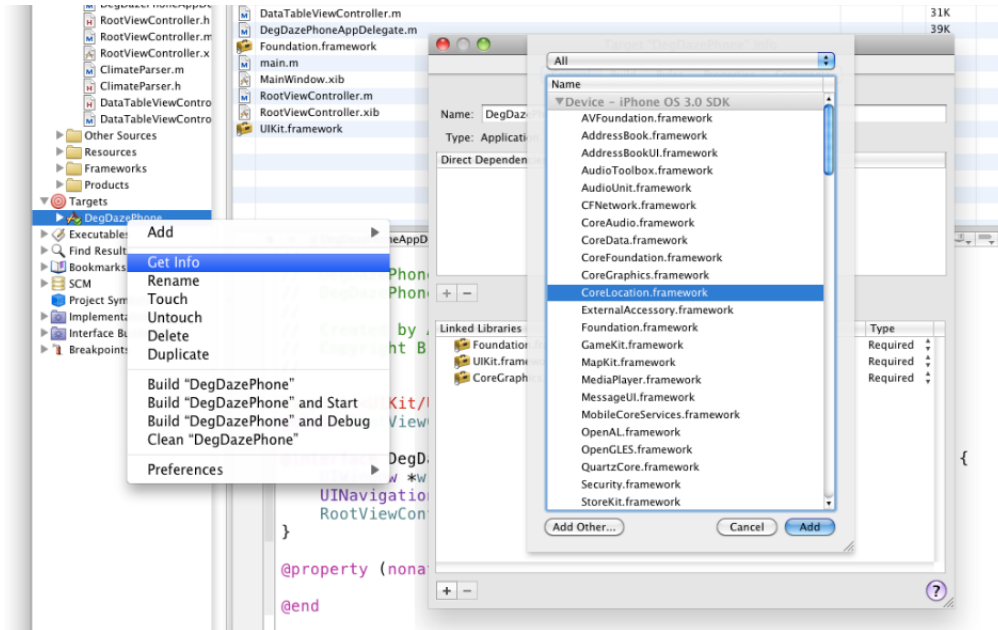
Core Location works the same on every iPhone OS device even though the underlying hardware might be different. Devices with a GPS unit can get more accurate location information, but in many contexts the less accurate cell tower or Wi-Fi methods work just fine. For example, our application does not need a very accurate location reading since climate tends to change gradually over a distance of many kilometers.

The Core Location framework consists of three objects: **CLLocationManager**, **CLLocation**, and **CLLocationHeading**. To get location information create an instance of **CLLocationManager** and assign one of your objects as its delegate.

CLLocationManager uses the hardware to produce the user's location and sends instances of **CLLocation** and **CLHeading** objects to it's delegate as they are updated.

Add the Core Location framework to the project.

Figure 2.12. Adding a framework



In `RootViewController.h` import the Core Location framework, declare that `RootViewController` implements the `CLLocationManagerDelegate` protocol, and add an instance variable to hold an instance of `CLLocationManager`.

```
#import <UIKit/UIKit.h>
#import <CoreLocation/CoreLocation.h>

@interface RootViewController : UIViewController <CLLocationManagerDelegate> {
    IBOutlet UITextField *latField;
    IBOutlet UITextField *lonField;
    IBOutlet UIButton *findMeButton;
    IBOutlet UIButton *fetchButton;
    IBOutlet UIActivityIndicatorView *progressIndicator;

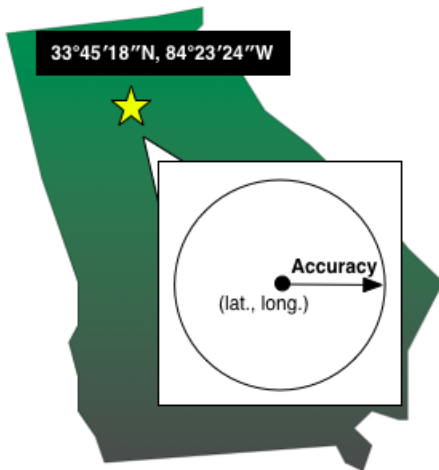
    NSMutableDictionary *climateRecords;

    // To find location
    CLLocationManager *locationManager;

    // For fetch
    NSMutableData *data;
    NSURLConnection *connection;
}
- (IBAction)findMeCancel:(id)sender;
- (IBAction)fetchCancel:(id)sender;
@end
```

When **CLLocationManager** is able to determine the user's location it sends a **CLLocation** object to it's delegate. The **CLLocation** object contains lots of information about the user's location, but the number of available properties depends on the available hardware of the device being used. All devices can obtain a user's latitude and longitude, but only devices with a GPS unit can return altitude and speed data. Along with the location coordinates **CLLocation** contains information on the horizontal accuracy of the reading.

Figure 2.13. Horizontal Accuracy



In `RootViewController.m` replace the `findMeCancel:` stub method, add a clean up method, and implement the `CLLocationManager` delegate methods.

```

- (void)cleanUpFindMe
{
    [locationManager release];
    locationManager = nil;
    [findMeButton setTitle:@"Find Me" forState:UIControlStateNormal];
    [progressIndicator stopAnimating];
}

- (IBAction)findMeCancel:(id)sender
{
    if (locationManager) {
        NSLog(@"canceling find me");
        [self performSelectorOnMainThread:@selector(cleanUpFindMe) withObject:nil
                                         waitUntilDone:NO];
    } else {
        // Create the location manager object and
        // set the root view controller as it's delegate
        locationManager = [[CLLocationManager alloc] init];
        [locationManager setDelegate:self];

        // Climate doesn't change much within many kilometers
        // so set the desired accuracy to the minimum
        [locationManager setDesiredAccuracy:kCLLocationAccuracyThreeKilometers];

        // Start looking for our location and
        // show the progress indicator
        [locationManager startUpdatingLocation];
        [findMeButton setTitle:@"Cancel" forState:UIControlStateNormal];
        [progressIndicator startAnimating];
    }
}

#pragma mark CLLocationManager delegate methods

- (void)locationManager:(CLLocationManager *)manager
  didUpdateToLocation:(CLLocation *)newLocation
  fromLocation:(CLLocation *)oldLocation
{
    CLLocationCoordinate2D latAndLong = [newLocation coordinate];

```

```
NSString *latString = [NSString stringWithFormat:@"%f", latAndLong.latitude];
[latField setText:latString];

NSString *lonString = [NSString stringWithFormat:@"%f", latAndLong.longitude];
[lonField setText:lonString];

[self performSelectorOnMainThread:@selector(cleanUpFindMe) withObject:nil
                               waitUntilDone:NO];
}

- (void)locationManager:(CLLocationManager *)manager
  didFailWithError:(NSError *)error
{
  NSLog(@"Could not find location: %@", error);
  [self performSelectorOnMainThread:@selector(cleanUpFindMe) withObject:nil
                               waitUntilDone:NO];
}
```

Build and run the application. You should now be able to get your latitude and longitude using Core Location and then fetch the climate record without ever using the keyboard.