

Extracted from:

# iPhone SDK Development

---

## Building iPhone Applications

This PDF file contains pages extracted from iPhone SDK Development, published by the Pragmatic Bookshelf. For more information or to purchase a paperback or PDF copy, please visit <http://www.pragprog.com>.

**Note:** This extract contains some colored text (particularly in code listing). This is available only in online versions of the books. The printed versions are black and white. Pagination might vary between the online and printer versions; the content is otherwise identical.

Copyright © 2009 The Pragmatic Programmers, LLC.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

The  
Pragmatic  
Programmers

## Add Book

Save

The  
Pragmatic  
Programmers

**iPhone SDK Development |**

Bill Dudney and  
Chris Adamson

Edited by Daniel H Steinberg





Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and The Pragmatic Programmers, LLC was aware of a trademark claim, the designations have been printed in initial capital letters or in all capitals. The Pragmatic Starter Kit, The Pragmatic Programmer, Pragmatic Programming, Pragmatic Bookshelf and the linking *g* device are trademarks of The Pragmatic Programmers, LLC.

Every precaution was taken in the preparation of this book. However, the publisher assumes no responsibility for errors or omissions, or for damages that may result from the use of information (including program listings) contained herein.

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at

<http://www.pragprog.com>

Copyright © 2009 Bill Dudney and Chris Adamson.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

Printed in the United States of America.

ISBN-10: 1-934356-25-5

ISBN-13: 978-1-934356-25-8

Printed on acid-free paper.

P2.0 printing, December 2009

Version: 2010-2-3

Location switch in IB or set the `showsUserLocation` property to YES, the map view will turn on the location manager and get updated with the current location. All we need to do is implement the proper map view delegate method. We will look at that in just a minute.

## 25.3 Map Annotations

There are two parts to each annotation on the map. The model piece of it is intended to be lightweight, so you can have many of them attached to a given map and not have to worry about the memory footprint. The other side is the view. Let's look at the model side first.

Map Kit defines the `MKAnnotation` protocol but no public implementations. So, in order to add an annotation to the map, we need to create our own implementation of this protocol. The protocol defines one property and two optional methods. The property is the location for the annotation; the methods are `title` and `subtitle`. The `title` is used as the text for the callout when the user clicks the annotation. The `subtitle` is displayed in smaller text under the title. Here is the header file for our annotation:

```
Download MapKit/ContactMapper_01/Classes/ContactAnnotation.h

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

@interface ContactAnnotation : NSObject <MKAnnotation> {
    CLLocationCoordinate2D _coordinate;
    NSString *_title;
    NSString *_subtitle;
    ABRecordRef _person;
}

+ (id)annotationWithCoordinate:(CLLocationCoordinate2D)coordinate;
- (id)initWithCoordinate:(CLLocationCoordinate2D)coordinate;

@property (nonatomic, assign) CLLocationCoordinate2D coordinate;
@property (nonatomic, assign) ABRecordRef person;
@property (nonatomic, copy) NSString *title;
@property (nonatomic, copy) NSString *subtitle;

@end
```

Since our annotation represents one of our contacts, we add a property named `person` for the contact. We also defined the `title` and `subtitle`

properties instead of just defining the methods defined in the protocol. Finally, we have the `initWithCoordinate:` and `annotationWithCoordinate:` methods that do as you'd expect, initialize a `ContactAnnotation`, and create and return an autoreleased annotation, respectively. Here is the implementation for the `ContactAnnotation`:

[Download](#) MapKit/ContactMapper\_01/Classes/ContactAnnotation.m

```
@implementation ContactAnnotation

@synthesize coordinate = _coordinate;
@synthesize title = _title;
@synthesize subtitle = _subtitle;
@synthesize person = _person;

+ (id)annotationWithCoordinate:(CLLocationCoordinate2D)coordinate {
    return [[[self class] alloc] initWithCoordinate:coordinate] autorelease];
}

- (id)initWithCoordinate:(CLLocationCoordinate2D)coordinate {
    self = [super init];
    if(nil != self) {
        self.coordinate = coordinate;
    }
    return self;
}

@end
```

Now that we have our annotation, we need to look at how we are going to create and add the annotation to the map. Thinking back to the way we want the application to work, when the user clicks the Choose button, we want to bring up the people picker. After a contact's address is chosen, we want to add the annotation. To make that happen, we have to do a couple of steps:

1. Add the `AddressBook` and `AddressBookUI` frameworks to the project.
2. Add an action method to the view controller, and implement it to present the person chooser.
3. Add a toolbar to the bottom of the view, and resize the map view to fit.
4. Connect the Choose button to the action method.
5. Implement the people picker delegate protocol to get the address when it's clicked.

Let's look at the action method. Here is the code:

[Download](#) MapKit/ContactMapper\_01/Classes/ContactMapperViewController.m

```
- (IBAction)choose {
    ABPeoplePickerNavigationController *picker =
    [[ABPeoplePickerNavigationController alloc] init];
    picker.peoplePickerDelegate = self;
    [self presentViewController:picker animated:YES];
    [picker release];
}
```

Now that we have the implementation, we need to add the declaration in the header file and then go back to IB to add the toolbar and make the connection from the button to the File's Owner.

For more detail on the people picker, see Chapter 23, *Address Book*, on page 445. Now that we have become the delegate of the people picker, we need to add that protocol to the header file and implement the methods. Of the three methods, we are going to look only at one, the `peoplePickerNavigationController:shouldContinueAfterSelectingPerson:property:identifier:method`. Here is the code:

[Download](#) MapKit/ContactMapper\_01/Classes/ContactMapperViewController.m

```
Line 1 - (BOOL)peoplePickerNavigationController:
- (ABPeoplePickerNavigationController *)peoplePicker
-     shouldContinueAfterSelectingPerson:(ABRecordRef)person
-                                     property:(ABPropertyID)property
5                                     identifier:(ABMultiValueIdentifier)identifier{
-     if(kABPersonAddressProperty == property) {
-         NSString *fullName = (NSString *)ABRecordCopyCompositeName(person);
-         CLLocationCoordinate2D coordinate = {0.0f, 0.0f};
-         self.newAnnotation = [ContactAnnotation annotationWithCoordinate:coordinate];
10        self.newAnnotation.title = fullName;
-         self.newAnnotation.person = person;
-         [fullName release];
-         ABMultiValueRef addresses =
-             ABRecordCopyValue(person, kABPersonAddressProperty);
15        CFIndex selectedAddressIndex =
-             ABMultiValueGetIndexForIdentifier(addresses, identifier);
-         CFDictionaryRef address =
-             ABMultiValueCopyValueAtIndex(addresses, selectedAddressIndex);
-         self.newAnnotation.coordinate = [AddressGeocoder locationOfAddress:address];
20        [self dismissModalViewControllerAnimated:YES];
-     }
-     return NO;
- }
```

If the contact's address was chosen, then we grab the contact's full name, create an annotation, set the `fullName` to be the title, and set the

annotation's person to be the selected contact. We also grab the address starting on line 13. We then geocode the address on line 19. We will look at the geocoding stuff in a moment (in Section 25.3, *TouchXML and the AddressGeocoder*, on page 476). After the user has chosen an address, we dismiss the person picker view.

Now our work is almost complete. Although we have created the new annotation, we have not added it to the map. We could do that in the people picker delegate method. However, we will get a much nicer animation if we place the code in the `viewDidAppear:` name. Here is the code:

Download MapKit/ContactMapper\_01/Classes/ContactMapperViewController.m

```
- (void)viewDidAppear:(BOOL)animated {
    [super viewDidAppear:animated];
    if(nil != self.newAnnotation) {
        [self.mapView addAnnotation:self.newAnnotation];
        self.newAnnotation = nil;
    }
    if(self.mapView.annotations.count > 1) {
        [self recenterMap];
    }
}
```

First we check to see whether we have a new annotation to add, and if so, we add it and then set the `newAnnotation` to `nil`. If we have more than one annotation, we also recenter the map by calling the `recenterMap` method. Recentering the map is straightforward; we build a new region centered between the annotations with a span that covers all the annotations. Here is the code:

Download MapKit/ContactMapper\_01/Classes/ContactMapperViewController.m

```
- (void)recenterMap {
    NSArray *coordinates = [self.mapView valueForKeyPath:@"annotations.coordinate"];
    CLLocationCoordinate2D maxCoord = {-90.0f, -180.0f};
    CLLocationCoordinate2D minCoord = {90.0f, 180.0f};
    for(NSValue *value in coordinates) {
        CLLocationCoordinate2D coord = {0.0f, 0.0f};
        [value getValue:&coord];
        if(coord.longitude > maxCoord.longitude) {
            maxCoord.longitude = coord.longitude;
        }
        if(coord.latitude > maxCoord.latitude) {
            maxCoord.latitude = coord.latitude;
        }
        if(coord.longitude < minCoord.longitude) {
            minCoord.longitude = coord.longitude;
        }
    }
}
```

```

        if(coord.latitude < minCoord.latitude) {
            minCoord.latitude = coord.latitude;
        }
    }
    MKCoordinateRegion region = {{0.0f, 0.0f}, {0.0f, 0.0f}};
    region.center.longitude = (minCoord.longitude + maxCoord.longitude) / 2.0;
    region.center.latitude = (minCoord.latitude + maxCoord.latitude) / 2.0;
    region.span.longitudeDelta = maxCoord.longitude - minCoord.longitude;
    region.span.latitudeDelta = maxCoord.latitude - minCoord.latitude;
    [self.mapView setRegion:region animated:YES];
}

```

After identifying the minimum and maximum latitude and longitude, we create a new region based on this min and max and then tell the map view to set its region to this new region.

Now we have an application that does most of what we want. Build and Go, and choose one of your contacts to see the pin drop and the map resize to fit your current location and the location of the chosen contact.

However, our pin is red, and we really wanted it to be purple. The default color is red, and since we have not done anything special, that is what we get. Although that works, we want a purple pin. To get a purple pin, we need to implement the MKMapViewDelegate protocol.

There are several methods in the MKMapViewDelegate protocol; most of them are callbacks to let you know when interesting stuff is happening with the map view (that is, `mapViewWillStartLoadingMap:`). The one we are interested in now, though, is the `mapView:viewForAnnotation:` method. Like the table view, the map view keeps a set of annotation views that can be dequeued and reused. Our implementation of this method needs to take that into account and make sure to use the dequeued view when possible. Here is the code:

**Download** MapKit/ContactMapper\_01/Classes/ContactMapperViewController.m

```

- (MKAnnotationView *)mapView:(MKMapView *)mapView
    viewForAnnotation:(id <MKAnnotation>)annotation {
    MKPinAnnotationView *view = nil;
    if(annotation != mapView.userLocation) {
        view = (MKPinAnnotationView *)
            [mapView dequeueReusableAnnotationViewWithIdentifier:@"identifier"];
        if(nil == view) {
            view = [[[MKPinAnnotationView alloc]
                initWithAnnotation:annotation reuseIdentifier:@"identifier"]
                autorelease];
        }
        [view setPinColor:MKPinAnnotationColorPurple];
    }
}

```



```

    [view setShowCallout:YES];
    [view setAnimatesDrop:YES];
} else {
    CLLocation *location = [[CLLocation alloc]
                           initWithLatitude:annotation.coordinate.latitude
                           longitude:annotation.coordinate.longitude];
    [self setCurrentLocation:location];
}
return view;
}

```

The first thing we do is check whether the current annotation is the current user location. We could return our own annotation view for the current user location, and it would work; however, if we return `nil`, then the default annotation view is used. The default annotation view for the current user's location is the blue dot that has the nice bouncing animation.

If this is not the user's current location, we dequeue an annotation view, and if there is not one to dequeue, we create one. Then we set up the annotation view the way we want it and then return the view. In our case, we are setting the pin color to purple, turning on callouts and specifying that the annotation should drop in with an animation.

If it is the user's current location, we call the `setCurrentLocation:`. This is where we get the Map Kit's integration with a location manager without having to implement the delegation methods ourselves.

## TouchXML and the AddressGeocoder

Now let's talk about the AddressGeocoder. The geocoder uses a web service to encode the addresses. We pass in an address and out pops XML with the address and the latitude and longitude. As you saw back in the Chapter 12, *Connecting to the Internet*, on page 246, we can use the `NSXMLParser` to parse XML, and although that works, it can be quite tedious to parse the whole XML file when you really want only a couple of bits of information. To get at the data we want, we have a couple of options. The iPhone includes the `libXML2` open source library, which works like a champ. But the API is entirely C-based, which makes it a bit of a cognitive disconnect when you'd rather be doing Objective-C. Enter the TouchXML open source library.

TouchXML provides an Objective-C wrapper over the `libXML2` library. The project is located on the Google Code website at <http://code.google.com/p/touchcode/wiki/TouchXML>. Version 1.0.6 is included in the code for this chapter, so you don't have to download it. The driver behind

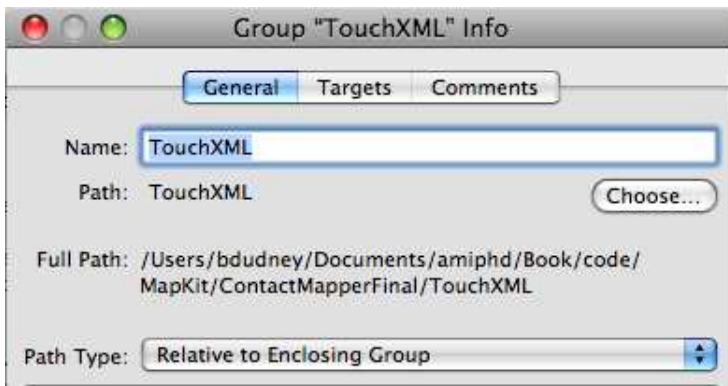
TouchXML is to provide the simplified XML APIs that are available on the Mac but not on the iPhone. So, if you want detailed documentation about one of the classes, you can replace the leading C with NS and look in the Mac documentation for details.

TouchXML provides us with XPath, which requires far less code to extract the information we need than doing the same with the parser. To get TouchXML working with your project, you need to do a couple of things:

- Get the code, either from the projects for this chapter or from the website.
- Add the source code for TouchXML and Tidy to your project.
- Add the libxml2 library to your linked libraries.
- Add the libxml2 header path to your header search path.

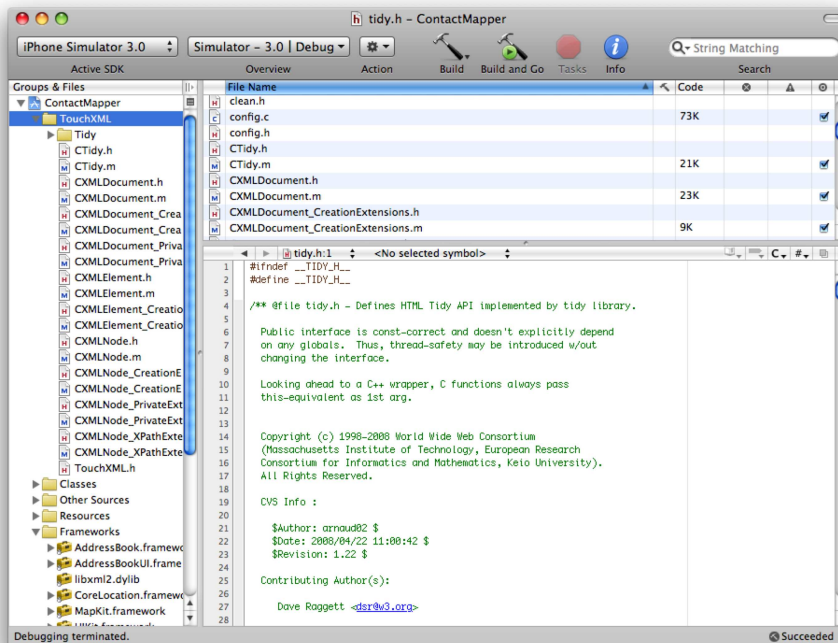
The TouchXML code includes the code for Tidy, so if you grabbed the code from TouchXML's website or from the code bundle for the book, you have the Tidy code. Let's look at adding the code to your project so you can use it instead of the parser.

First we need to add a group for TouchXML and then add a group under TouchXML for Tidy. Select the project, Ctrl+click, and choose Add > New Group. Name the group TouchXML. Select the group, right-click, and choose Get Info. On the General tab, click the Choose button near the Path item. Create a new folder called TouchXML, select it, and hit the Choose button. Repeat the steps for the Tidy group, select the TouchXML group, right-click, choose Add > New Group, name it Tidy, and specify its path is the Tidy folder. Here is a screenshot of the TouchXML folder in our project:

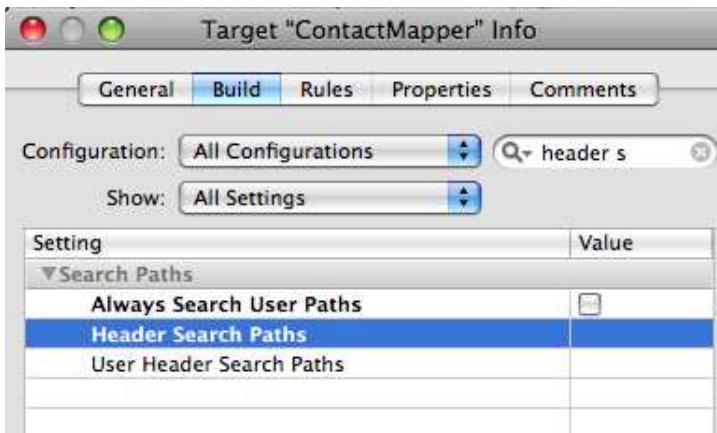


Depending on where you got the code, the directory structure will be different. If you are grabbing the code from the book's source bundle, then everything is under the TouchXML directory and the Tidy directory under that. If you are grabbing the source from the TouchXML project site, the TouchXML code is under Common/Source and Common/Source/Creation. The Tidy source is under Externals/tidy/src and Externals/tidy/include. You need all the files from both TouchXML and Tidy.

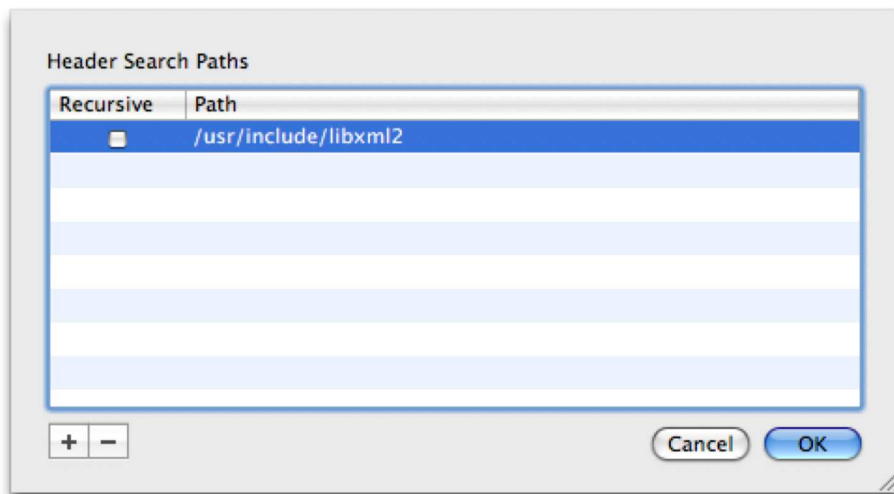
In the Finder, select all the files for TouchXML, and drag them into the TouchXML group in Xcode. When prompted, make sure to select the “copy files” checkbox. Do the same for the Tidy files, dragging them into the Tidy group, and make sure to copy them as well. When you are done, Xcode should look roughly like this:



Now that we have all the code, we need to tell Xcode where to find the libXML2 headers. Select your target under the Targets group. Right-click or ^-click the item, and choose Get Info. On the Build tab, type header s into the search field, and you should see something like the next screenshot.



Double-click the line that reads Header Search Paths. In the sheet that pops up, click the + button, and type `/usr/include/libxml2` into the Path column. When you are done, it should look like this:



Click the OK button, and you should be able to build with TouchXML and Tidy as part of your project.

There are other ways to approach this; we could instead make a static library from the TouchXML and Tidy source, but that is beyond what we will cover.

Now that we have TouchXML set up and running, let's take a look at the AddressGeocoder class and how it uses TouchXML to get an address back from one of the geocoding services.

# The Pragmatic Bookshelf

---

The Pragmatic Bookshelf features books written by developers for developers. The titles continue the well-known Pragmatic Programmer style and continue to garner awards and rave reviews. As development gets more and more difficult, the Pragmatic Programmers will be there with more titles and products to help you stay on top of your game.

## Visit Us Online

---

### **iPhone SDK Development's Home Page**

<http://pragprog.com/titles/amiphd>

Source code from this book, errata, and other resources. Come give us feedback, too!

### **Register for Updates**

<http://pragprog.com/updates>

Be notified when updates and new books become available.

### **Join the Community**

<http://pragprog.com/community>

Read our weblogs, join our online discussions, participate in our mailing list, interact with our wiki, and benefit from the experience of other Pragmatic Programmers.

### **New and Noteworthy**

<http://pragprog.com/news>

Check out the latest pragmatic developments, new titles and other offerings.

## Buy the Book

---

If you liked this eBook, perhaps you'd like to have a paper copy of the book. It's available for purchase at our store: [pragprog.com/titles/amiphd](http://pragprog.com/titles/amiphd).

## Contact Us

---

Online Orders:	<a href="http://www.pragprog.com/catalog">www.pragprog.com/catalog</a>
Customer Service:	<a href="mailto:support@pragprog.com">support@pragprog.com</a>
Non-English Versions:	<a href="mailto:translations@pragprog.com">translations@pragprog.com</a>
Pragmatic Teaching:	<a href="mailto:academic@pragprog.com">academic@pragprog.com</a>
Author Proposals:	<a href="mailto:proposals@pragprog.com">proposals@pragprog.com</a>
Contact us:	1-800-699-PROG (+1 919 847 3884)