

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 © 2008 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.

When the user's finger touches the screen over a view the view is sent the `touchesBegan:withEvent:` method. If the view is a button or other control it processes the event by invoking its actions, if the view is one of your views and you don't implement the method then the default implementation on `UIView` is invoked and the responder chain is sent the message until either something responds or the event is discarded. As the user moves the touch around on the screen a series of `touchesMoved:withEvent:` methods are sent and are processed in the same way. When the user lifts their finger(s) off the screen the `touchesEnded:withEvent:` is sent and again is processed by the responder chain. If the system interrupts the application (because of a system event like an incoming phone call) the application is sent the `touchesCancelled:withEvent:` to give your application the chance to clean up any info it was keeping related to the current event processing happening.

In each of the event methods the first argument is a set of the touches that have changed since the last event method was called. The second argument is the event that contains all the touches (via the `allTouches` property) or all the touches for a particular view (via `allTouchesForView:`). Each touch knows which view it started in and its current and previous location in that view.

Implementing these event processing methods and using the touches information is key to making your application multi-touch. In the remainder of this chapter we will be looking at the details of how to make applications do many of the cool multi-touch things that other iPhone apps do like swipe and rotate. Throughout the remainder of this chapter we are going to implement several examples of handling events. Let's get started with tracking touches.

10.2 Tracking Touches

To provide a good multi-touch experience to your users you have to understand how to track touches. As the user's finger moves across elements the direction and duration are used to discern intent in interesting ways. For example when you touch an email cell in the mail application and slide your finger across the cell the Delete button appears so you can delete the email. This event type is known as a swipe. In the Photos application 'flicking' takes the user to the next or previous photo, pinching and stretching cause the photo to zoom in or out. Tracking touches is what makes multi-touch gestures possible.

Enough talk about events lets dig into some code. In this example we are going to build an application that tracks our touches and draws dots where we have touched. Figure 10.2, on the following page shows the example application with my name drawn out.

To make this application work the view tracks both the beginning of the touch as well as the touch moving. As with all event streams the touches begin in the `touchesBegan:withEvent:` method. Here is the code.

[Download](#) MultiTouch/Draw/Classes/DrawView.m

```
Line 1 - (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
-     UITouch *touch = [touches anyObject];
-     CGPoint location = [touch locationInView:self];
-     Dot *dot = [[[Dot alloc] init] autorelease];
5     dot.x = location.x;
-     dot.y = location.y;
-     [self.dots addObject:dot];
-     [self setNeedsDisplay];
- }
```

In this code we are creating the first Dot for the sequence of events. On line 3 we get the location of the touch in the view and then create a Dot for that location and add that to the list and then mark the view for display. The Dot class is very simple and is only a holder of the x and y values.

The list of Dot's are used to draw circles (i.e. the dots) in the view, one for each instance at the location captured in the object. Keep in mind that this code is meant to give us a visual of what tracking our events can do for us, not an example of how to do the most efficient drawing on the iPhone.

Next is the `touchesMoved:withEvent:` method. Here is the code.

[Download](#) MultiTouch/Draw/Classes/DrawView.m

```
- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event {
-     UITouch *touch = [touches anyObject];
-     CGPoint location = [touch locationInView:self];
-     Dot *dot = [[[Dot alloc] init] autorelease];
-     dot.x = location.x;
-     dot.y = location.y;
-     [self.dots addObject:dot];
-     [self setNeedsDisplay];
- }
```

In this method we are doing more or less the same thing as in the `touchesBegan:withEvent:` except that this method is called many more times than the previous method.



Figure 10.2: Dots from Touches

10.3 Tapping

Tapping is another fundamental way users interact with their iPhones. Taps go through the same cycle of `touchesBegan`, `touchesMoved` and `touchesEnded` as other events so in order to make our applications respond to one or more taps we implement one or more of these methods. In the simple case where we want to provide a 'double tap' action for one of our responders we can implement `touchesBegan:withEvent:`. If the `tapCount` property is equal to two then we invoke the 'double tap' functionality. If not then we ignore the event.

In the previous example we saw one way we can respond to a multi tap event. In the example we cleared all the dots out of the view when the user double taps. The drawing view only responds to a double tap. An interesting case comes up if we want a responder to do one action on a single tap and do a different action on a double tap. The interesting part is that the system will send the responder the first single tap event and then send the second tap as a new event. The responder gets what looks like a single tap and a double tap.

Turns out that making this work is not difficult but it can be confusing if you've not done something like it before. The trick is to delay the performance of the single tap action until you know if it's really a single tap or the first of a double tap sequence. The way we do that on the iPhone is with the `performSelector:withObject:afterDelay:` method. This method queues up the execution of a method (a selector is the data structure the ObjC runtime uses to represent methods) to happen some time later.

Once we queue up the response to the initial single tap we can then wait for the double tap and if it does come we can cancel the initial single tap invocation via another method called `cancelPreviousPerformRequestsWithTarget:`. All this can seem a bit abstract, let's look at the code. In this example the view turns red on a single tap and turns blue on a double tap. Here is part of the code.

[Download](#) MultiTouch/MultiTap/Classes/MultiTapView.m

```
- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event {
    UITouch *touch = [touches anyObject];
    if(touch.tapCount == 1) {
        [self performSelector:@selector(turnRed) withObject:nil afterDelay:0.10f];
    }
    if(touch.tapCount == 2) {
        [self turnBlue];
    }
}
```

```
}
```

When the tap ends (the user picks up their finger) and the tapCount is 1 the single tap method is queued up to be executed in 0.1 seconds. If the tapCount is 2 then the double tap method is invoked right away. This is relatively straightforward code. However we are not canceling the single tap method so if this was all we had then both would happen and the view would always be red. Let's look at the code to fix that.

[Download](#) MultiTouch/MultiTap/Classes/MultiTapView.m

```
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
    UITouch *touch = [touches anyObject];
    if(touch.tapCount == 2) {
        [[self class] cancelPreviousPerformRequestsWithTarget:self
            selector:@selector(turnRed)
            object:nil];
    }
}
```

If the second tap comes in then the single tap method is canceled. The delay we put in of 0.1 seconds gives the user that amount of time to tap a second time and register the double tap event, otherwise it will be another single tap. Using this technique we can give multiple actions to our responders. For example you might want to provide a control that selects on a single tap but edits on a double tap.

10.4 Multi-Touch Gestures

To this point we have been looking at scenarios where we consider only one finger down on the screen at a time. While that covers most of the event handling needed for most applications there are cases where more than one finger can be used to indicate more complex interactions. The canonical case in the built-in apps is the pinch gesture. It is hard to imagine a better user interaction model that could be done with one finger, who wants to tap buttons even if they fade in and out or whatever when the two finger gesture is so natural. We should strive for the same type of natural interaction model in our apps. Make something amazing that the users already know how to use.

Let's go back to the simple draw application we did back in Section 10.2, *Tracking Touches*, on page 145 and modify it to draw dots for each touch on the screen. It is surprisingly simple to add multi-touch drawing to this application. The first thing we need to do is turn on multi-touch for the view. We could do this in code by setting the multiple-

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 in the news.

Buy the Book

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

Contact Us

| | |
|-----------------------|--|
| Phone Orders: | 1-800-699-PROG (+1 919 847 3884) |
| Online Orders: | www.pragprog.com/catalog |
| Customer Service: | orders@pragprog.com |
| Non-English Versions: | translations@pragprog.com |
| Pragmatic Teaching: | academic@pragprog.com |
| Author Proposals: | proposals@pragprog.com |