Chapter 7

Accessing People, Places, Objects, and Relationships

In this chapter, we’ll cover the nuts and bolts of Facebook methods, objects, properties, and connections—and how to get at them. We’ll also introduce JSON, or JavaScript Object Notifications, which are ancillary to the use of the Graph API. Finally, we’ll talk about retrieving basic data from Twitter’s REST (Representational State Transfer)1 API.

You can find all of the code for this chapter in the Chapter7 directory of the Git repository. The Facebook code is in the ApiFacebook project, and the Twitter code is in the ApiTwitter project. These projects build off the same application structure that was introduced in the Chapter 6’s sample projects; and once again, the projects aren’t pretty, but they get the job done.

More Fun with the Facebook Graph API

In the last chapter, we showed you how to pull information from Facebook’s social graph. As you did this, you were probably left wondering how to go about adding or posting information from your own app to Facebook’s social graph. Well, since we’re such nice guys, we’ve gone through the trouble of dedicating an entire section of this chapter to posting to the Facebook social graph. We’ve also added a thorough review of additional information that you can pull from the social graph, including how that information relates to authorization and extended permissions. Read on for the gory details.

1 See, for example, http://en.wikipedia.org/wiki/Representational_State_Transfer
Facebook Dialogs

One of the great ways to spice up your iOS application and make it a hit with users is to let them post to their Facebook page directly from within your app. Even though iOS supports copy and paste and fast switching between apps, users won’t find your app appealing if they have to switch to the iOS Facebook app itself to, for instance, post a link to an interesting article from within your application to their Facebook wall.

Fortunately for us, the Facebook SDK has made it as simple as possible to get up and running with this functionality. This brings us to the dialog methods in the Facebook class that we have yet to discuss:

- (void)dialog:(NSString *)action
  andDelegate:(id<FBDialogDelegate>)delegate;

- (void)dialog:(NSString *)action
  andParams:(NSMutableDictionary *)params
  andDelegate:(id <FBDialogDelegate>)delegate;

Both of these methods are in Facebook.h; and while there are two methods available to us, we will focus on using the second one, which lets us pass in additional parameters. The first method without parameters is usable, but more often than not you will need to pass parameters to the dialog: method. Moreover, if you look in Facebook.m, you will see that the first method calls the second method with an empty dictionary for the parameters:

- (void)dialog:(NSString *)action
  andDelegate:(id<FBDialogDelegate>)delegate {
    NSMutableDictionary * params = [NSMutableDictionary dictionary];
    [self dialog:action andParams:params andDelegate:delegate];
}

Both of these methods also take an action parameter and a delegate parameter. We will look at these now in our sample application. In the sample application for this chapter, we have a new class entitled DialogViewController. This class will look awfully similar to the LoginViewController class because, lo and behold, it’s modeled directly after it. That said, we want to focus our attention on a few things within the DialogViewController class.

Since we are going to be displaying dialogs to the user from the DialogViewController class, we need to declare that it is an FBDialogDelegate in our header file, DialogViewController.h:

@interface DialogViewController : UIViewController <FBDialogDelegate> {
}
@end

In DialogViewController.m, it’s up to us to define each of the following delegate callback methods:

- (void)dialogDidComplete:(FBDialog *)dialog;
- (void)dialogCompleteWithUrl:(NSURL *)url;
- (void)dialogDidNotCompleteWithUrl:(NSURL *)url;
- (void)dialogDidNotComplete:(FBDialog *)dialog;