MySpaceID iPhone SDK
04.22.2010

This document outlines how to get started using the MySpaceID iPhone SDK. The SDK is meant to support OffSite and OnSite applications that wish to make applications on the iPhone. The SDK will enable your application to login MySpace users and access their approved data. You will have access to all the REST V1, OpenSocail 0.9, and OpenSearch APIs currently supported on the MySpace Developer Platform.

OAuth on iPhone

Due to the particularities of the OAuth specification, users are required to input their credentials at the OAuth provider. This requires iPhone applications to launch the Safari browser and send their user to the MySpace Authentication page. Once the user input his/her valid credentials, the OAuth spec requires the provider to return to the callback URL that was used when coming to the auth page. On the iPhone, we are able to register an application's URL scheme, which will permit other applications to launch your app. The iPhone SDK encapsulates this entire process for you, making it easy to set up 3-legged-OAuth on your iPhone app. You will need to define a unique URL scheme that you believe other apps will not use, in order to avoid confusion on the iPhone. The OAuth libraries are largely based upon work done by Jon Crosby, with some tweaking needed to accommodate our requirements. Special thanks to him.

Parsing the Http Responses

We recognize that developers would like to have their prerogative in choosing the parsing mechanism they prefer. The previous version of the iPhone SDK, XML was the response format for most of the API calls. Due to the difficult nature of XML parsing on the iPhone, I have included sample code that enables XPath queries on XML data. This is C code that utilizes the libxml2.2.dylib available on the iPhone. I've specifically left XML parsing code out of the SDK classes so that developers can choose their own method of parsing. For more details regarding the XPath code, please refer to Matt Gallagher's blog post Using libxml2 for XML parsing and XPath queries in Cocoa.

For JSON parsing, we have included JSON.Framework into the project for parsing. The JSON folder contains and compiles all that is necessary for JSON parsing to work. Note that there is coupling between the RoaApi (OpenSocial 0.9) and the JSON.Framework code due to the requirement that Json code be added to an HTTP Request Body for updates to the API. Therefore, JSON.Framework is a necessary piece of the SDK.

In this version of the SDK, we introduced a MSApi class which encapsulates both the API library and MSSecurityContext that is used in the underlying code. It is meant to

simplify developer adoption. If you are interested in using the underlying code, be our guest. You can refer to the unit tests to get a better idea of how those classes work.

Preparing your MySpaceID Application

If your application is a MySpaceID application (OffSite), then before you can use the iPhone SDK, you must set the correct External Callback URL to the URL Scheme you plan to use for your iPhone Application. To do this, go to your application settings page, and find the section named "External Site Settings". The external callback validation url must have this form: YOUR_URL_SCHEME://oauthcallback/. In the case of the sample application, the external callback URL would be myspaceid://oauthcallback/. See the next section for more on URL schemes.

Using the SDK

1. Your application needs to handle the URL scheme that you set for it. Adding an applicationHandleOpenURL method to your application delegate can do this. See MySpaceID_DemoAppDelegate.m in the demo for the details.
2. You need to register a unique URL Type within your .plist file. This UrlScheme must match the external callback URL scheme mentioned above.
3. Make sure to set the UrlScheme property of the context to the same value you are using in the .plist when you instantiate the MySpace class. This will allow the SDK to create an appropriate callback URL that will fire off your application when the user has authenticated. PLEASE DO NOT USE THE DEFAULT URLSCHEME.
4. Code your applications knowing that the user could be logged on or off. You can check if the user is logged in by calling [mySpace isLoggedIn]. This checks if your app has an Access Token associated with the user for Offsite applications (Onsite applications do not need an Access Token, hence checking if they are logged in is mute).
5. Request and Access tokens are being stored within your applications UserDefaults via NSUserDefaults. This permits the SDK to access the tokens wherever in the code in needs it.
6. When instantiating the MSApi class, ensure that you set the IsOnsite flag to indicate whether your application is an OpenSocial Onsite app or a MySpaceID Offsite app.
7. To use in another project, simply copy the MySpaceID folder to your project. Feel free to remove the XPathQuery folder if you do not plan to use the XML parser included. We recommend you use the JSON parser as we've noticed it performs faster.
8. The MSApi class can operate in synchronous and asynchronous mode. It is recommended that you use asynchronous mode because network latency will lock your UI main thread causing your application to freeze until the

network call completes. However, there are cases where you may need a synchronous call, so we left that as an option.

   a. To use MSApi asynchronously, pass a delegate into the constructor method. That delegate has to implement the MSRequest protocol. The method names of the called API will be returned to the delegate method once it completes. You can see a sample of this in the PersonViewController.
   b. To use MSApi synchronously, do not pass a delegate into the constructor. Note: getRequest Token and getAccessToken shall remain synchronous by default.

9. Every API method is given an optional NSDictionary parameter named "queryParams". This is designed to allow developers to add additional query parameters into the request. Since OpenSocial is an evolving and large spec, this dictionary parameter is meant to allow developers to use the OpenSocial specification's optional functionality. Please review the OpenSocial implementation within the MySpace Developer Platform to find out what optional parameters are currently supported. There are helper methods for creating typical NSDictionary objects that are currently supported within MySpace. You can find these methods (named "makeQueryDictionary") in the MSOpenSearch, MSRestV1 and MSRoaApi classes.

Using MySpaceID.Demo

This sample comes with two view controllers that help get you familiar with the SDK. The first controller is meant for unit testing. It lists all the major APIs supported per endpoint library. In order to make use of the UnitTestViewController, make sure to go to the setupContext method and enter the correct consumer_key, consumer_secret, access_token_key and access_token_secret of your application. Otherwise any tests that require OAuth will fail. You can run all the tests or click them individually. If you are debugging the application, you can see the responses in the console window. You will also see the reason why unit tests fail.

The PersonViewController demonstrates how to make simple get calls, update your status, and to upload a media item (photo or video) for your **MySpaceID** application. Please note that this controller is set to use the SDK for Offsite applications, so when you are testing, make sure you are testing with a MySpaceID (offsite) application. The first time you go to this tab, you will see a Login button in the navigation bar; click this to begin the login process. Note that if you did not correctly setup your MySpaceID application and your URLSchemes as described above, the Login process will fail.

The UnitTestViewController utilizes the underlying API classes and use synchronous requests, although each individual unit test runs in its own thread. The PersonViewController makes use of the MSApi wrapper class, and utilizes the API in Asynchronous mode. It is recommended that developers take the PersonViewController as a model for SDK usage.