Introduction to Facebook APIs

Write Facebook apps for the Android platform with the Facebook SDK for Android

Skill Level: Introductory

C. Enrique Ortiz
Developer and author
About Mobility Weblog

14 Dec 2010

Updated 16 Dec 2010

You can incorporate Facebook functionality into your own applications. From the mobile perspective, the Facebook Platform supports APIs for mobile web applications, and mobile SDKs for native mobile applications for the iPhone, iPad, and Android platforms. In this article, explore the Facebook Platform APIs and the Facebook SDK for Android, the SDK released by the Facebook mobile team.

16 Dec 2010 - *Per request of author, refreshed the download file and changed the filename to fb-sampleapp.zip (see Download).*

Frequently used acronyms

- ADT: Android Development Tools
- API: Application program interface
- IDE: Integrated development environment
- JDK: Java Development Kit
- JSON: JavaScript Object Notation
- REST: Representational State Transfer
- SDK: Software Development Kit
- UI: User interface

- URL: Uniform Resource Locator
- XML: Extensible Markup Language

Prerequisites

To follow along with this article, you need the following skills and tools:

- Basic knowledge of Java™ technology and how to use Eclipse (or your favorite IDE)
- Java Development Kit (version 5 or 6 required)
- Eclipse (version 3.4 or 3.5)
- Android SDK and ADT plug-in

For download and setup information, see Resources at the end of this article.

Overview of the Facebook Platform APIs

Before covering the Facebook SDK for Android, it is important to understand the Facebook Platform and its APIs. According to Facebook, the Facebook Platform allows anyone to "build social applications on Facebook and the Web." To enable you to build such applications, Facebook offers an extensive collection of core and advanced APIs and SDKs (see Resources).

The core Facebook Platform API is the Graph API that allows you to read and write data to and from Facebook. Facebook also has what is called the Old Rest API. The newer Graph API changes the API paradigm from a method-oriented way of reading and writing data to and from Facebook to a new way that uses objects (think user profiles, friends, posts, photos, likes, and so on) and their relationships or connections with each other. This approach simplifies the Facebook API and makes it more consistent when working with objects. Note that while the Graph API is the preferred Facebook API, the Old REST API is still active and supported. Both the Graph and the REST APIs are applicable to mobile applications, both native and mobile web applications, including mobile web content within native applications through the use of WebViews.

Graph API objects are assigned a unique ID and are easily addressable using a URL that can be further qualified to address a specific object/connection. The general structure of an object URL is as follows: `https://graph.facebook.com/OBJECT_ID/CONNECTION_TYPE` where `OBJECT_ID` is the object's unique ID and `CONNECTION_TYPE` is one of the

connection types supported by the object. For example, a page supports the following connections: feed/wall, photos, notes, posts, members, and so on.

With the Graph API, you can retrieve an object, delete an object, and publish objects. You can search, update objects, filter results, and even dynamically discover the connections/relationships of an object.

By default, applications have access to the user's public data. To access private data, applications must first request the user's permissions, which are called extended permissions. Facebook defines a large number of permissions that you can read at the Extended Permissions page (see Resources).

Introduction to the Facebook SDK for Android

Now that you have a better understanding of the Facebook Platform APIs, look at the Facebook SDK for Android.

The Facebook SDK for Android is a Java programming language wrapper to the Facebook Graph and old REST APIs. This SDK is open source, and it is hosted at github's facebook / facebook-android-sdk repository (see Resources). Note that due to the evolving nature of the open-source SDK, expect future changes to the SDK. The SDK is licensed under the Apache License, Version 2.0.

The Facebook SDK for Android hides a lot of the details covered in the previous section, Overview of the Facebook Platform APIs. It does so by providing six Java classes as described in Table 1.

Table 1. Package com.facebook.android

Classes	Description
`AsyncFacebookRunner`	A helper class that implements asynchronous Facebook API calls
`DialogError`	A class that encapsulates dialog error
`Facebook`	Main Facebook class for interaction with the Facebook Platform APIs
`FacebookError`	A class that encapsulates a Facebook error
`FbDialog`	A class that implements a WebView for Facebook dialogs
`Util`	A helper class with a number of utility methods

The Facebook SDK for Android also brings a couple of useful examples that you can use as a baseline for your own applications.

Of special interest are the core `Facebook` class and the `Facebook Dialog` class,

which I cover in more detail next. The core `Facebook` class encapsulates methods to authorize the user, create Facebook dialogs, make API requests, log out the user, and get or set access and session information and status. The `Facebook Dialog` class implements a WebView and methods for its creation as well as the logic for handling Facebook URL (status) responses. The dialog is central to the way the SDK operates. The SDK provides two methods to authenticate, one that is referred to as single-sign on which uses, if installed, the native Facebook application dialogs, and the default WebView approach to dialogs. In this article, I will focus on the WebView approach. The rest of the SDK classes are helper classes used to encapsulate error information or provide useful utilities used throughout the SDK.

The following sections focus on use-cases for a typical Facebook application:

- Installing the Facebook SDK for Android
- Registering your application
- Creating the SampleApp
- Displaying Facebook dialogs
- Authorizing the user
- Making API requests

Installing the Android SDK

You have to download and install Eclipse or your favorite IDE. In addition, you must install the Android SDK. For information on how to download and install Eclipse and the Android SDK, see the Download the Android SDK page at the Android Developer web site (see Resources).

Installing the Facebook SDK for Android

The Facebook SDK for Android is hosted at github's facebook / facebook-android-sdk repository. Download the SDK source, and extract it into your working directory. Start Eclipse, and create an Android Project by selecting **File Menu -> New -> Project** and choose the Android Project. Create the project from the `existing source` and point to the extracted source directory (see Download for the source code).

Registering your application

Before you start, you have to register your Facebook application and get an

application ID (`client_id`) and associated secret key (`client_secret`). The client ID is used in your application when making the different Facebook API calls.

The sample application

The sample application, which for simplicity sake I call SampleApp, shows you how to use the Facebook SDK for Android. The application consists of a single file that implements the SampleApp activity and that uses one screen to display messages, the list of Facebook friends and menu items to login/authenticate the user, get a list of friends, and post to the authenticated user's (`me`) wall.

Before you start, remember that you must register your application with Facebook, as previously explained, and that you have to set the attribute `APP_ID` in SampleApp.java to the `client_id` as provided by Facebook (see Listing 1).

Listing 1. Initializing the application ID

```
// Set application ID to your registered app client_id
// See http://www.facebook.com/developers/createapp.php
public static final String APP_ID = ".....";
```

For the Sample Application screen, use a linear layout that contains a TextView for simple status messages, a ListView to display the list of Facebook friends retrieved from the server, and menu items to logic/authenticate the user, get a list of friends, and post to the authenticated user's (me) wall. Listing 2 shows the XML UI declarations for the main UI screen layout.

Listing 2. Main screen UI declarations

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@drawable/black"
    >

    <TextView android:id="@+id/txt"
        android:text="@string/hello"
        android:textColor="@drawable/white"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:paddingRight="10dp"
        android:paddingLeft="10dp"
        android:layout_margin="10dp"
        android:textSize="10sp"
        android:layout_marginTop="5px"
        android:layout_marginBottom="5px"
        android:layout_marginRight="0px"
        android:layout_marginLeft="0px"
        android:gravity="left"
```

```
    />

    <ListView
        android:id="@+id/friendsview"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textFilterEnabled="true"
    />
    <TextView
        android:id="@id/android:empty"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="No data"
     />
</LinearLayout>
```

Listing 3 shows the XML UI declarations for each row in your ListView.

Listing 3. ListView row UI declarations

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="?android:attr/listPreferredItemHeight"
    android:padding="10dip"
    >
     <TextView
         android:id="@+id/rowtext_top"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:gravity="center_vertical"
     />
</LinearLayout>
```

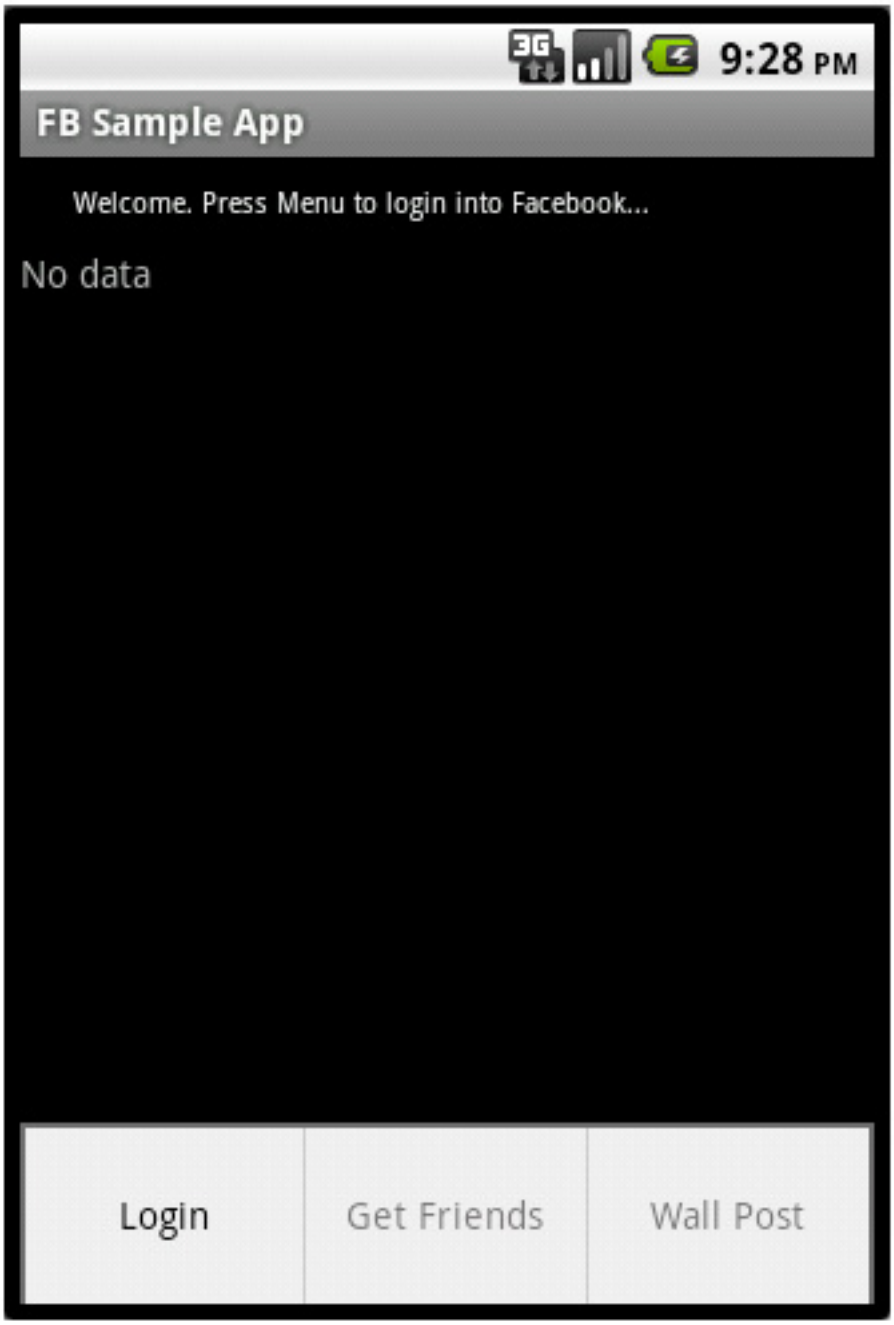
And Listing 4 shows the XML UI declarations for the menu items.

Listing 4. Menu items UI declarations

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/login" android:title="Login"/>
    <item android:id="@+id/getfriends" android:title="Get Friends"/>
    <item android:id="@+id/wallpost" android:title="Wall Post" />
</menu>
```

Figure 1 shows the results of the XML UI declaration for SampleApp before (with the Login button) and after login (with the Logout, Get Friends, and Wall Post buttons).

Figure 1. SampleApp screen layout



The SampleApp `onCreate()` method is invoked by the Android platform when the SampleApp instance is created. This method performs a basic check to make sure that the App ID is set before proceeding, initializes the UI resources, and initializes the Facebook session instance (see Listing 5).

Listing 5. Initializing the application

```
/** Called when the activity is first created. */
```

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Make sure the app client_app has been set
    if (APP_ID == null) {
        Util.showAlert(this,
        "Warning", "Facebook Application ID must be set...");
    }

    // Initialize the content view
    setContentView(R.layout.main);
    // Get the status text line resource
    mText = (TextView) SampleApp.this.findViewById(R.id.txt);

    // Setup the ListView Adapter that is loaded when selecting "get friends"
    listView = (ListView) findViewById(R.id.friendsview);
    friendsArrayAdapter = new FriendsArrayAdapter(this, R.layout.rowlayout, friends);
    listView.setAdapter(friendsArrayAdapter);

    // Define a spinner used when loading the friends over the network
    mSpinner = new ProgressDialog(listView.getContext());
    mSpinner.requestWindowFeature(Window.FEATURE_NO_TITLE);
    mSpinner.setMessage("Loading...");

    // Initialize the Facebook session
    mFacebook = new Facebook(APP_ID);
    mAsyncRunner = new AsyncFacebookRunner(mFacebook);

}
```

The rest of the application flow is triggered by UI interactions through the menu items. Three menu items are defined 1) login/logout (toggle), 2) get friends, and 3) post to the wall, as illustrated in Listing 4. As the user selects a menu item, the application performs the appropriate action. The following listing shows how the menu items are handled. Listing 6 shows how to instantiate the menu by deflating its XML definition.

Listing 6. Creating the menu

```
/**
 * Invoked at the time to create the menu
 * @param the menu to create
 */
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.main_menu, menu);
    return true;
}
```

Listing 7 shows how the menu items are modified before getting displayed, that is, show the proper "login" or "logout" state and enable/disable "get friends" and "post to wall" as appropriate.

Listing 7. Creating the menu

```
/**
 * Invoked when preparing to display the menu
 * @param menu is the menu to prepare
 */
@Override
public boolean onPrepareOptionsMenu(Menu menu) {
    MenuItem loginItem = menu.findItem(R.id.login);
    MenuItem postItem = menu.findItem(R.id.wallpost);
    MenuItem getfriendsItem = menu.findItem(R.id.getfriends);
    if (mFacebook.isSessionValid()) {
        loginItem.setTitle("Logout");
        postItem.setEnabled(true);
        getfriendsItem.setEnabled(true);
    } else {
        loginItem.setTitle("Login");
        postItem.setEnabled(false);
        getfriendsItem.setEnabled(false);
    }
    loginItem.setEnabled(true);
    return super.onPrepareOptionsMenu(menu);
}
```

Listing 8 shows how the menu items are handled and the appropriate action (login/logout, get friends, post to wall) invoked.

Listing 8. Handling the menu selection

```
/**
 * Invoked when a menu item has been selected
 * @param item is the selected menu item
 */
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {

        // Login/logout toggle
        case R.id.login:
            // Toggle the button state.
            //  If coming from login transition to logout.
            if (mFacebook.isSessionValid()) {
                AsyncFacebookRunner asyncRunner = new AsyncFacebookRunner(mFacebook);
                asyncRunner.logout(this.getBaseContext(), new LogoutRequestListener());
            } else {
                // Toggle the button state.
                //  If coming from logout transition to login (authorize).
                mFacebook.authorize(this, PERMISSIONS, new LoginDialogListener());
            }
            break;

        // Wall Post
        case R.id.wallpost: // Wall Post
            mFacebook.dialog(SampleApp.this, "stream.publish", new
                                                    WallPostDialogListener());
            break;

        // Get Friend's List
        case R.id.getfriends: // Wall Post
            // Get the authenticated user's friends
            mSpinner.show();
            mAsyncRunner.request("me/friends", new FriendsRequestListener());
            break;

        default:
            return false;
```

```
    }
    return true;
}
```

The application uses a number of callbacks to handle the state of each asynchronous call. The SampleApp defines a callback to handle the success or failure for login requests, for logout dialogs, for post to wall dialogs and for get friends requests. You can take this sample application and add your own actions by just following the same pattern of adding menus and the corresponding calls to Facebook with the corresponding callbacks to receive state messages.

Both the login and "post to wall" menu items invoke the Facebook dialog which by default uses the SDK WebView-based dialogs if the single-sign on option is not defined; this is explained in the following section.

The `FriendsRequestListener` callback is covered in Making Facebook API Requests. The last callback, `MyDialogListener` implements the Facebook SDK for Android `DialogListener`, which is used for Facebook requests that uses the SDK WebView-based dialogs, as explained in the following section. The implementer of `DialogListener` must implement the appropriate completion logic; for example, this is used during login and wall post requests.

Displaying Facebook dialogs

The Facebook SDK for Android relies on Facebook web/server-based dialogs for user interactions such as user authorization, permissions, and message publishing. The core SDK `Facebook.java` class defines the method `dialog()` when it generates the UI dialog for the request action (see Listing 9).

Listing 9. The Facebook Dialog method

```
/**
 * Generate a UI dialog for the request action in the given Android context
 * with the provided parameters.
 */
public void dialog(
        Context context,
        String action,
        Bundle parameters,
        final DialogListener listener) {
    :
    :
}
```

This asynchronous method takes for input the application context, the action to perform (such as `login`, `publish_stream`, `read_stream`, `offline_access`), the required parameters for the specific request or action, and a listener method that is called when the asynchronous methods complete executing. The method creates

the appropriate WebView dialog. For example, to publish a status message, you make a call to the core Facebook class method `dialog()` passing the action `stream.publish` (see Listing 10).

Listing 10. Calling the dialog() method and processing the callback

```
package com.myapp.facebook.android;
:

import com.facebook.android.*;
:

mFacebook = new Facebook(); // Facebook core
:

// Create a Facebook dialog (using the call asynchronous API)
mFacebook.dialog(
    this,
    "stream.publish",
    new MyDialogListener());

:
:

//
// My asynchronous dialog listener
//
public class MyDialogListener extends com.facebook.android.Facebook.DialogListener {

    public void onComplete(Bundle values) {

        final String postId = values.getString("post_id");

        if (postId != null) {

            // "Wall post made..."

        } else {
            // "No wall post made..."
        }

    }

    public void onFacebookError(FacebookError e) {...}
    public void onError(DialogError e) {...}
    public void onCancel() {...}

}
```

The result of this call is the Facebook web-based **publish to wall** dialog as in Figure 2.

Figure 2. Posting to the wall

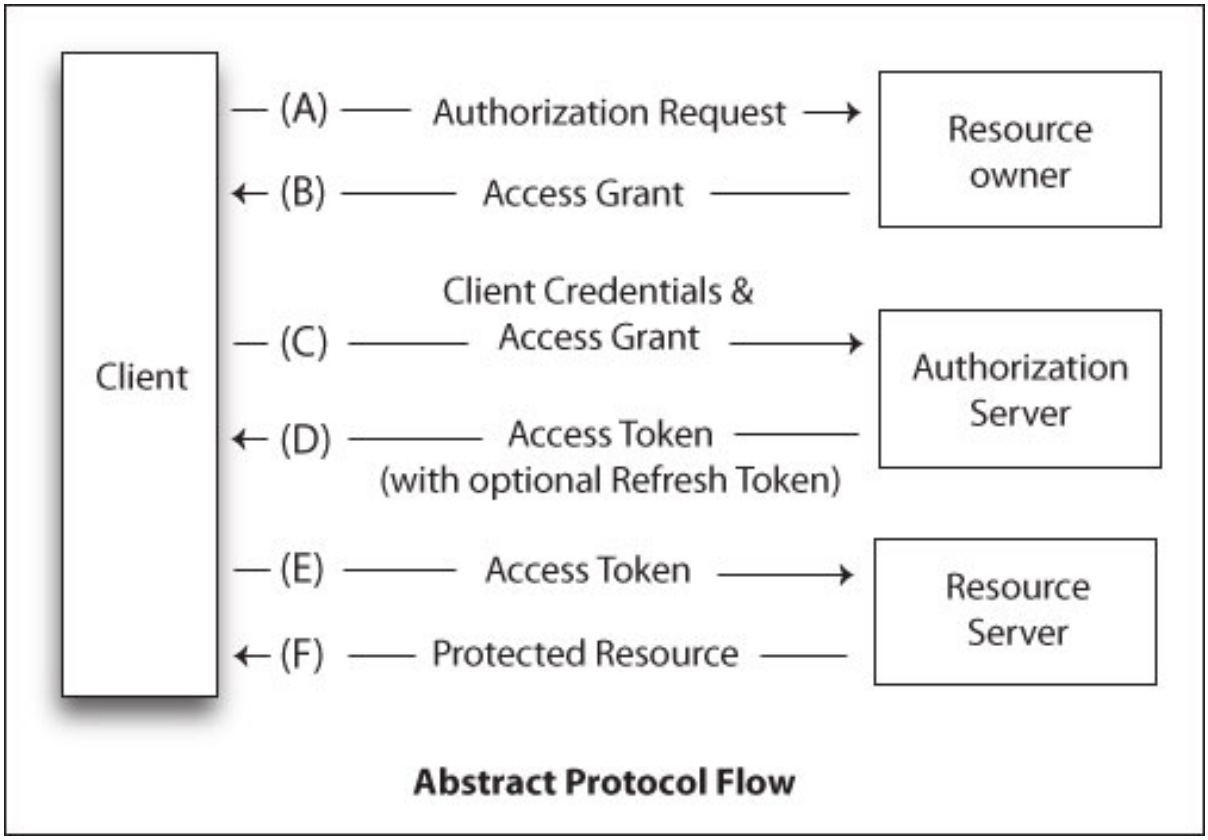
This is the familiar Web page for posting to the user's wall as served by Facebook. The user can skip or go ahead and publish the message. This use of native and web content together within the same mobile application is a great example of the power of hybrid applications.

Authorizing users

The Facebook Platform provides support for OAuth 2.0 authentication as well as support for an older, custom authorization signature scheme. You should avoid using the older authentication scheme when writing new applications as support for this older scheme will not be available in the near future. For more information about OAuth, see the OAuth 2.0 Protocol specification in Resources.

The Facebook SDK for Android together with the Facebook Platform hides the complexity of OAuth authentication as illustrated in Figure 3.

Figure 3. The OAuth 2.0 Protocol (IETF)



The Facebook SDK for Android takes a mobile web approach to authentication rather than a native one. Note that it is possible that future versions of the SDK will provide native Android support for OAuth. For this approach, the SDK uses the Facebook web-based authentication dialog inside a WebView. This is an interesting approach for a couple of reasons:

- It reduces the complexity of implementing OAuth native to the application.
- Probably as important, it wins the user's trust by displaying the same standard and familiar login and permission dialogs as seen when using Facebook on regular browsers.

One disadvantage of this approach is that it might feel slower sometimes than a pure

native implementation. For this, the SDK includes an alternative authentication/login approach that uses the native Facebook application's single sign-on. This is, if the official Facebook native application is installed on the handset, the Facebook SDK for Android can use its authorization/login Activity. But to make this work, you must first sign your application (see Frank Ableson's article, "Introduction to Android Security" in Resources), then generate a signature or key hash (see Listing 11).

Listing 11. Generating the application key hash

```
keytool -exportcert -alias [alias] -keystore [keystore] | openssl sha1 -binary |
                                                                    openssl base64
```

Note that the code in Listing 11 normally appears on a single line. It is split to multiple lines for formating purposes.

You must then register the generated key hash on Facebook for your particular application under the mobile and devices section as in Figure 4. (View a larger version of Figure 4.)

Figure 4. Enter the application key hash

Once registered, you must define an "activity result handler" for your application; implement `onActivityResult(...)` on which you must invoke the method `facebook.authorizeCallback(...)` as in Listing 12).

Listing 12. Defining an activity result handler

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
  super.onActivityResult(requestCode, resultCode, data);
  facebook.authorizeCallback(requestCode, resultCode, data);
  :
  :
  // ... Other onActivityResult steps per your app...
}
```

Note that if the single sign-on method is not defined or the native Facebook application is not present, the SDK will default to the WebView approach for authentication/login. The rest of the article focuses on this default approach.

As previously mentioned, by default an application has access to all public data in the user's profile. To access private data, the application must request and the user

must grant to the application permissions to do so. The permissions to gain access to private data are referred to as Extended Permissions.

The Facebook SDK for Android `authorize()` method implements the OAuth 2.0 User-Agent flow to retrieve an access token for use in subsequent API requests. The method starts either an Activity (native Facebook app, if present and configured) or a the WebView dialog to prompt the user for credentials (authentication) and permissions (authorization), see Listing 13).

Listing 13. The authorize method

```
/**
 * Authorize method that grants custom permissions.
 */
public void authorize(Activity activity, String[] permissions,
        final DialogListener listener) {
    :
    :
}
```

Note that for the single sign-on authentication to properly work, you must include a call to the `authorizeCallback()` method in your `onActivityResult()` function.

Recall from Listing 8 how the login menu item invokes the `authorize()` method with permissions.

Listing 14 shows how to invoke the `authorize()` method with permissions to write content. It also has permission to write comments and likes (`publish_stream`), read the user's feeds, perform searches (`read_stream`), and make the access credentials long-lived (`offline_access`). Don't forget to set your `APP_ID`.

Listing 14. Call the authorize() method with permissions

```
package com.myapp.facebook.android;
:
import com.facebook.android.*;
:

public static final String APP_ID = "13234...";

private static final String[] PERMISSIONS = new String[] {"publish_stream",
        "read_stream", "offline_access"};

:
:

// Call the authorization method
mFacebook.authorize(
    getContext(),
    APP_ID,
    mPermissions,
    new MyLoginDialogListener());
```

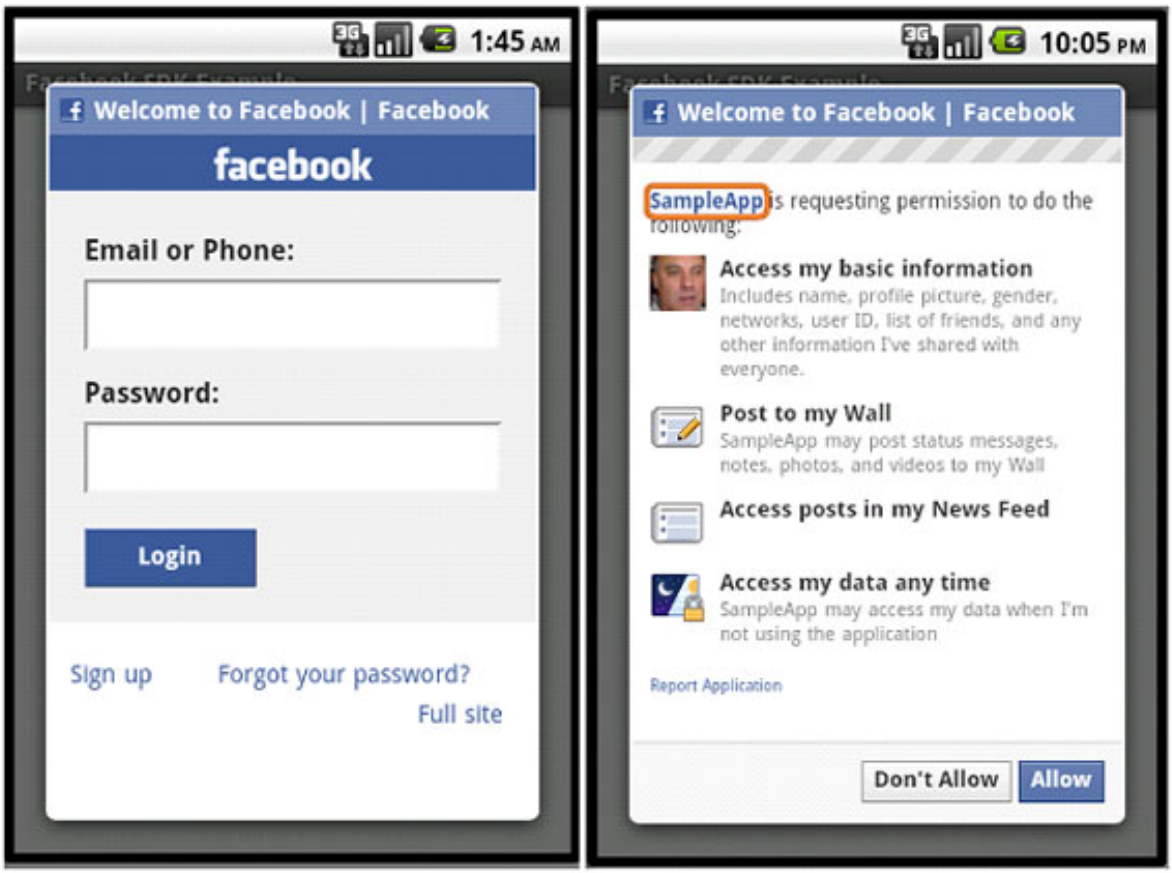
```
:
:

// Asynchronous Callback when authorization completes
private final class MyLoginDialogListener implements com.facebook.android
          .Facebook.DialogListener {

    public void onComplete(Bundle values) {...} // here enable logout
    public void onFacebookError(FacebookError error) {...}
    public void onError(DialogError error) {...}
    public void onCancel() {...}
}
```

The `authorize()` method internally invokes the `dialog()` method previously covered, but this time it passes a Login request for action. With the help of the Facebook Platform and its implementation of OAuth, the `authorize()` method also takes care of the authentication request to Facebook and returns the access_token that is used throughout the Facebook session and API invocation: `https://graph.facebook.com/ID?access_token=...`.

For detailed information on how authentication works, see the Facebook Authentication page in Resources.

Calling the `authorize()` method results in the WebView-based dialogs in Figure 5.

Figure 5. The authorize and permissions screens (WebView)



After the user logs in, she is prompted for permissions (email or phone information and a password) as requested by the application. The user can allow or deny the application permission to access her Facebook and do such actions as access basic information, post to the user's Wall, access posts in the News Feed, or Access user data.

Making Facebook API requests

The core Facebook class implements a number of `request()` methods for Old REST and Graph API calls. In addition, the SDK provides helper wrapper classes to make the core request API calls asynchronous.

As an example of an API request, you will make a request to retrieve the friends for the authenticated user. Recall the structure of a Graph API:

```
https://graph.facebook.com/ID/CONNECTION_TYPE
```

Also, recall the special ID `me` that is used to identify the authenticated user; the user has logged in: `https://graph.facebook.com/me/friends`.

Listing 15 shows a snippet from Listing 8 on how to make the `request()` method call. In this example, you use the `AsyncFacebookRunner.request(...)` method, which is a helper wrapper method to make the underlying request method asynchronous by running in its own thread. This step is done to avoid blocking the main UI thread.

Listing 15. Dispatching a request asynchronously

```
// Get Friend's List
case R.id.getfriends: // Wall Post
    // Get the authenticated user's friends
    mSpinner.show();
    mAsyncRunner.request("me/friends", new FriendsRequestListener());
    break;
```

The `FriendsRequestListener onComplete(...)` callback is invoked when the request to get the friend's list finishes executing. This method parses the returned JSON for the list of friends. Note that Listing 16 does not show all of the implemented methods.

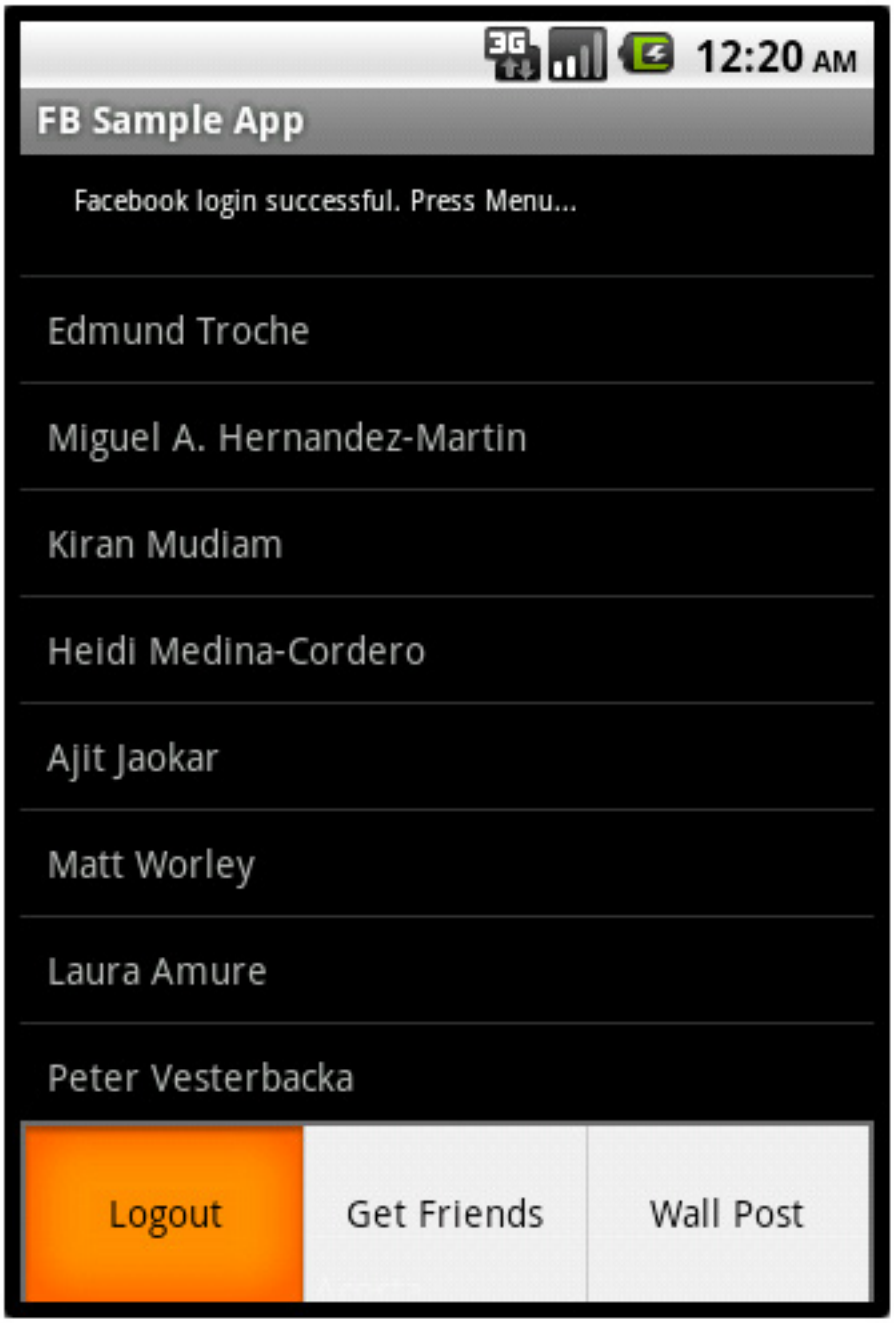
Listing 16. Processing the get friends request

```
/**
 * FriendsRequestListener implements a request lister/callback
 *  for "get friends" requests
 */
public class FriendsRequestListener implements
        com.facebook.android.AsyncFacebookRunner.RequestListener {

    /**
     * Called when the request to get friends has been completed.
     * Retrieve and parse and display the JSON stream.
     */
    public void onComplete(final String response) {
        mSpinner.dismiss();
        try {
            // process the response here: executed in background thread
            Log.d("Facebook-Example-Friends Request", "response.length(): " +
                                                        response.length());
            Log.d("Facebook-Example-Friends Request", "Response: " + response);

            final JSONObject json = new JSONObject(response);
            JSONArray d = json.getJSONArray("data");
            int l = (d != null ? d.length() : 0);
            Log.d("Facebook-Example-Friends Request", "d.length(): " + l);

            for (int i=0; i<l; i++) {
                JSONObject o = d.getJSONObject(i);
                String n = o.getString("name");
                String id = o.getString("id");
                Friend f = new Friend();
                f.id = id;
                f.name = n;
                friends.add(f);
            }

            // Only the original owner thread can touch its views
            SampleApp.this.runOnUiThread(new Runnable() {
                public void run() {
```

```
                    friendsArrayAdapter = new FriendsArrayAdapter(
                            SampleApp.this, R.layout.rowlayout, friends);
                    listView.setAdapter(friendsArrayAdapter);
                    friendsArrayAdapter.notifyDataSetChanged();
                }
            });
        } catch (JSONException e) {
            Log.w("Facebook-Example", "JSON Error in response");
        }
    }

    .
    .
}
```

According to your application requirements, the response from the Facebook servers, which is in JSON format, must be parsed and processed accordingly. You might decide to display such information by using a WebView or by using a ListView as in this SampleApp (see Figure 6).

Figure 6. Displaying the results



Conclusion

In this article, you looked at Facebook APIs. You began with a general overview of the Facebook Platform and its APIs, followed by the Facebook SDK for Android and a sample application. With this information, you now have a better understanding of the different Facebook APIs that are available and of how to start writing Facebook applications for the Android platform.

Downloads

Description	Name	Size	Download method
Article source code	fb_sampleapp.zip	12KB	HTTP

Information about download methods

Resources

Learn

- Facebook mobile web page: Learn how to incorporate Facebook into your own mobile application using the same APIs as those provided for all websites, formatted to fit on a mobile phone.
- Create an Application - Facebook Developers: Register your Facebook application.
- Extended permissions - Facebook Developers: Request extended permissions if your application needs to access other parts of the user's profile that might be private, or if your application needs to publish content to Facebook on a user's behalf.
- The official Facebook developer documentation: Explore the powerful APIs that enable you to create social experiences to drive growth and engagement on your website.
- Facebook Authentication: Learn about authentication and authorization when using the Facebook platform to develop your applications.
- The Facebook Developer Roadmap: Use this roadmap to plan for changes that might require code modifications.
- Develop Android applications with Eclipse (Frank Ableson, developerWorks, February 2008): The easiest way to develop Android applications is to use Eclipse. Learn all about this topic in this developerWorks tutorial.
- Introduction to Android development (Frank Ableson, developerWorks, May 2009): Get an introduction to the Android platform and learn how to code a basic Android application.
- Networking with Android (Frank Ableson, developerWorks, June 2009): Explore the networking capabilities of Android.
- Working with XML on Android (Michael Galpin, developerWorks, June 2009): Learn about the different options for working with XML on Android and how to use them to build your own Android applications.
- Under the Hood of Native Web Apps for Android: Learn about hybrid applications in Android.
- Unlocking Android (Frank Ableson, Manning Publications, 2010): Cover all aspects of Android development in this book.
- Mobile Design and Development (Brian Fling, O'Reilly Media, 2009): Read about practical guidelines, standards, techniques, and best practices for building mobile products in this book.

- Android SDK documentation: Get the latest information in the Android API reference.
- Documentation for the `android.webkit` package online: Find more information on setting up the WebView control.
- The Open Handset Alliance: Visit Android's sponsor.
- More articles by this author (C. Enrique Ortiz, developerWorks, July 2004-current): Read articles about Android, mobile applications, web services, and other technologies.
- XML area on developerWorks: Get the resources you need to advance your skills in the XML arena.
- My developerWorks: Personalize your developerWorks experience.
- IBM XML certification: Find out how you can become an IBM-Certified Developer in XML and related technologies.
- XML technical library: See the developerWorks XML Zone for a wide range of technical articles and tips, tutorials, standards, and IBM Redbooks. Also, read more XML tips.
- developerWorks technical events and webcasts: Stay current with technology in these sessions.
- developerWorks on Twitter: Join today to follow developerWorks tweets.
- developerWorks podcasts: Listen to interesting interviews and discussions for software developers.
- developerWorks on-demand demos: Watch demos ranging from product installation and setup for beginners to advanced functionality for experienced developers.

Get products and technologies

- Facebook SDK for Android (Currently an Alpha release): Work with this library to integrate Facebook into your Android mobile application.
- The Facebook Platform: Expand your ability to build social applications on Facebook and the web.
- The Facebook Old REST API (The previous version of the Graph API): Interact with the Facebook website programmatically through simple HTTP requests.
- The Facebook Graph API (The current version): Dig into this core Facebook Platform API.
- OAuth 2.0 Protocol specification (July 2010): Work with OAuth authentication, supported by the Facebook Platform.

- Android SDK: Download the SDK, access the API reference, and get the latest news on Android from the official Android developers' site. Version 1.5 and later will work.
- The Android Open Source Project: Find the open source information and source code you need to build an Android-compatible device.
- JDK 6 Update 21: Get the Java Platform, Standard Edition.
- Eclipse: Obtain the latest Eclipse IDE.
- The Facebook SDK for Android: Get the current Alpha release of the Facebook SDK for Android.
- IBM product evaluation versions: Download or explore the online trials in the IBM SOA Sandbox and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

Discuss

- XML zone discussion forums: Participate in any of several XML-related discussions.
- The developerWorks community: Connect with other developerWorks users while exploring the developer-driven blogs, forums, groups, and wikis.

About the author

C. Enrique Ortiz

C. Enrique Ortiz is a long-time mobile technologist, developer and author. He blogs at the About Mobility weblog and is the founder of the Austin chapter of MobileMonday.

Trademarks

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.
Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
Other company, product, or service names may be trademarks or service marks of others.