Mobage Native SDK Programming Guide

iOS Version (Mobage West)

This guide explains the basics of developing games with the Mobage Native SDK for iOS. We will walk you through step-by-step to integrate the SDK into a new or existing project and make the first Mobage API call.

Note: This guide may have been updated after the Native SDK was released. The most recent version of the Programming Guide is always available on the Mobage Developer Portal: https://developer.mobage.com/en/resources?set_sdk=native

Change History

Ver.	Changes	Last Update
2.1	Mobage Native SDK 2.1 release.	2012/11/07
2.0.1	Removed information about reporting lifecycle events, which is no longer required in version 2.0 and later.	2012/10/01
2.0	Mobage Native SDK 2.0 release.	2012/09/06
1.5 Hotfix	Updated requirement on Bundle ID.	2012/06/05
1.5	Mobage Native SDK 1.5 release.	2012/04/09
1.5-preview	Mobage Native SDK 1.5-preview release.	2012/03/30

Table of Contents

1. Overview

Mobage is a powerful social gaming platform. Building your app with the Mobage Native SDK gives you access to the functions and services of the Mobage platform.

Integrating the Mobage Native SDK into your iOS applications consists of these main tasks. The following pages have more detail about these tasks.

1. Create a new account and an app on the Mobage Developer Portal.

   https://developer.mobage.com/

   You will need to upgrade your account (free of charge) by agreeing to our Mobage legal agreements before you can create apps. For more details, please refer to the documentation available on the Mobage Developer Portal.

2. Create a new iOS app project in Xcode and import the Mobage Native SDK into it, or directly import the SDK into your existing project. See page 6 for details.

3. Call the APIs provided by the Mobage Native SDK to bring more fun to your game! See page 12 for details.

Important: **All apps must be published by Mobage.** You must not submit your apps to the App Store directly. All apps must be uploaded through the Mobage Developer Portal instead.

Important: **You must use Mobage's pre-defined bundle identifier to build your game.** The bundle identifier is in the form `com.mobage.ww.aXXX.hellomobageios`, where "aXXX" is an internal code generated by the Mobage Developer Portal. You can find this on the Details page of your iOS app in the Mobage Developer Portal after you click the Prepare for Submission button. If you have already used a different bundle identifier, ensure that you update your project's info.plist file.

2. The Mobage Native SDK for iOS

This section provides details about the development environment required to develop with the Mobage Native SDK, as well as the supported device operating systems. Later, this section discusses the files that are bundled with the SDK.

Development environment

Your development environment needs to meet the following requirements to use the Mobage Native SDK for iOS.

- **[OS]** OS X 10.7 or later.
- **[Development Tools]** Xcode 4.5 or later. **Note**: As of version 2.1, earlier versions of Xcode are no longer supported.
- **[Target Device OS]** iOS 5 or later. **Note**: As of version 2.1, iOS 4.3.3 is no longer supported.

Environment used in this guide

- **[OS]** Mac OS X Lion (10.7.4)
- **[Xcode]** Xcode 4.5
- **[Target]** iOS 5.0.1

Contents of the Mobage Native SDK for iOS

The Mobage Native SDK for iOS contains the following files:

File/Folder name	Description
README.txt	An important file that you should read on every release of the SDK.
US/	
UPGRADE.txt	Instructions for upgrading from earlier versions of the Mobage Native SDK.
Documentation/	
ProgrammingGuide_iOS_en.pdf	This document.
API Docs/	The API reference documentation that describes all classes and API methods in detail.
Samples/	
HelloMobage/	The “Hello Mobage” app shown in this document.
NDK/	
release/	
MobageNDK.framework/	The main framework file that provides all Mobage API methods.
NDKResources.bundle	The main resource bundle file that contains all other Mobage resources, such as images.

You will learn how to use these files in later sections.

3. Using the Mobage Native SDK for iOS

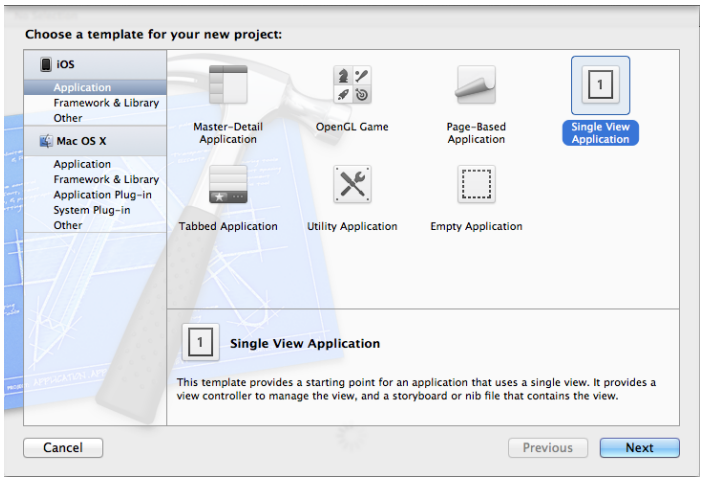
To start development with the Mobage Native SDK, add the Mobage Native SDK to an iOS project in Xcode. In this section, we will learn how to create a new project in Xcode. If you already have experience with iOS development in Xcode, please skip to “Importing the Mobage Native SDK” on page 8.

A sample Xcode project created based on these instructions is included in the SDK for your reference.

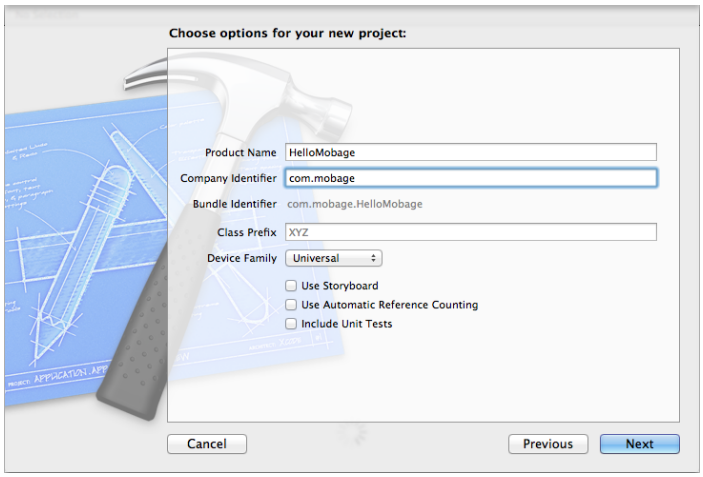
Creating a new iOS project in Xcode

When you launch the Xcode IDE, a welcome window is displayed. Select “Create a new Xcode project” to create a project.

If this dialog does not show up, click File > New > New Project …



The project template selection window is then displayed. Please select the template that you want to use. Here, we use the “Single View Application” template. This will create a simple iOS app that only has one view.



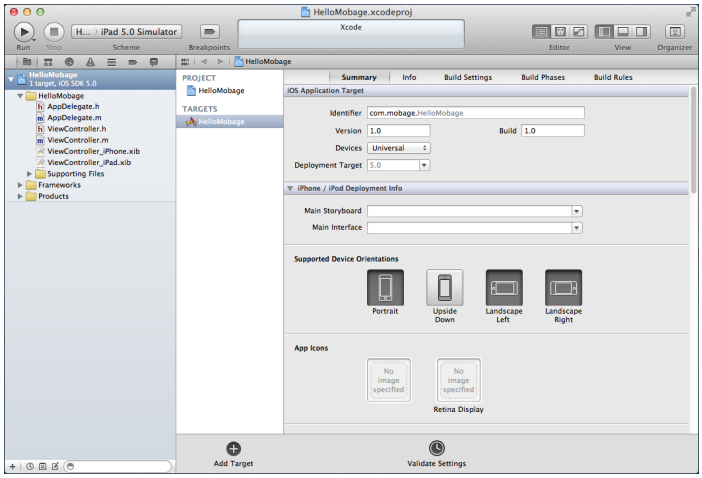
Next, you are asked to enter details about your app, such as the app's name, your company identifier, etc. Please see the note below regarding the Product Name and Company Identifier.

For the details of these values, please refer to Apple's documentation.

Click Next.

Important: **You must use Mobage's pre-defined bundle identifier to build your game.** The bundle identifier is in the form `com.mobage.ww.aXXX.hellomobageios`, where "aXXX" is an internal code generated by the Mobage Developer Portal. You can find this on the Details page of your iOS app in the Mobage Developer Portal after you click the Prepare for Submission button. Since Xcode's wizard does not allow you to specify the bundle identifier directly, you may want to use a dummy bundle identifier here and update it later in your project's info.plist file.

Finally, Xcode prompts you for the location to save your project. Select the destination folder, then click Create.



If you see a window similar to the left image, your project has been created successfully.

Importing the Mobage Native SDK

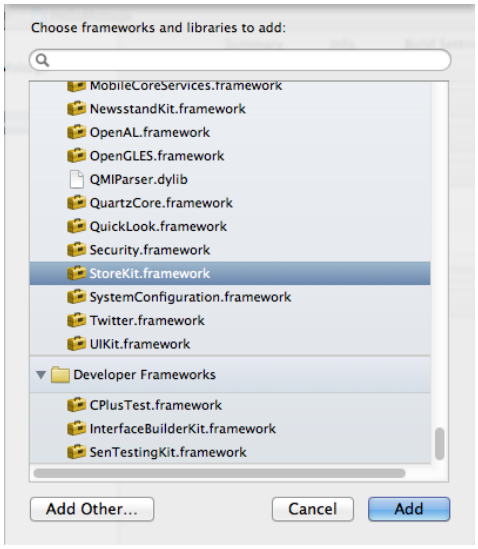
To import the Mobage Native SDK into your project, complete the following steps:

i) Adding standard frameworks and libraries

Before adding the Native SDK framework to your project, you must add several standard frameworks and libraries provided by iOS. The Native SDK framework (MobageNDK.framework) depends on the following frameworks and libraries to function.

- AdSupport.framework (set the linking to “Optional”)
- CoreGraphics.framework
- CoreTelephony.framework
- Foundation.framework
- libsqlite3.dylib
- MessageUI.framework
- QuartzCore.framework
- Security.framework
- StoreKit.framework
- SystemConfiguration.framework
- Twitter.framework
- UIKit.framework

To add these frameworks and libraries to your project:



Click on the Project icon (“HelloMobage”) at the top of the left pane.

Select the target (“HelloMobage”) from the TARGETS list in the middle pane.

On the right side of the middle pane, select the Build Phases tab.

Expand the Link Binary With Libraries section.

Click the “+” button at the bottom of the expanded section.

Add the required frameworks and libraries into your project by selecting them from the list and clicking Add. Use Command+Click to select multiple items.

ii) Adding the Native SDK framework

After you have added the standard frameworks and libraries to your project, you can add the Mobage Native SDK framework (MobageNDK.framework):

Click on the Project icon (“HelloMobage”) at the top of the left pane.

Select the target (“HelloMobage”) from the TARGETS list in the middle pane.

On the right side of the middle pane, select the Build Phases tab.

Expand the Link Binary With Libraries section.

Click the “+” button at the bottom of the expanded section.

Click Add Other… to locate the Native SDK framework, then click Add.

iii) Adding the Native SDK resource bundle

After adding the required frameworks and libraries, you can add the Native SDK’s resources bundle to your project:

Click on the Project icon (“HelloMobage”) at the top of the left pane.

Select the target (“HelloMobage”) from the TARGETS list in the middle pane.

On the right side of the middle pane, select the Build Phases tab.

Expand the Copy Bundle Resources section.

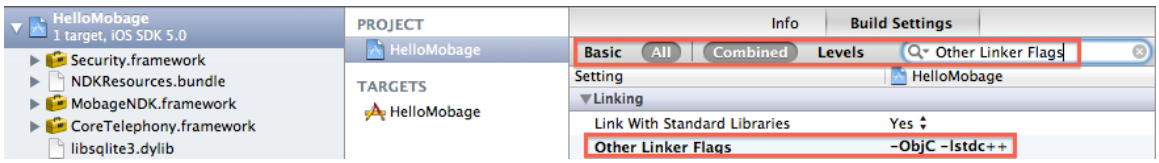
Click the “+” button at the bottom of the expanded section.

Click Add Other… to locate the Native SDK resource bundle, then click Add.

iv) Adding extra linker flags

Finally, to ensure that your build uses the linker flags required by the Native SDK, update your project build settings to include the following flags:

-ObjC -lstdc++



Additional configuration

After you import the Native SDK into your Xcode project, you must add a URL scheme to your app's build target and update the project's localization settings.

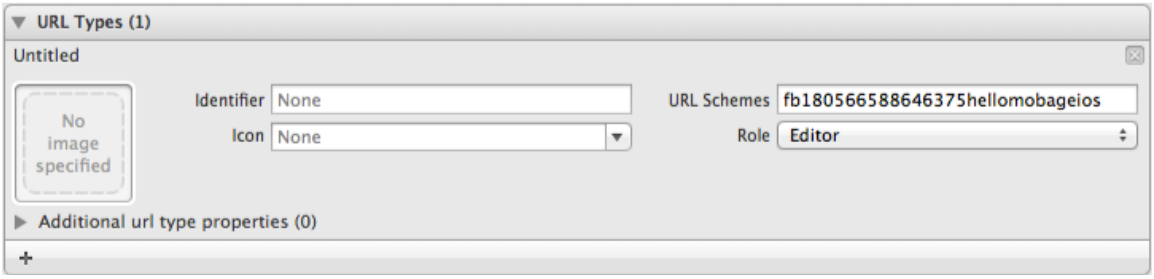
i) Adding the URL scheme

The Native SDK includes support for Facebook Connect, which allows your users to sign into Mobage using an existing Facebook account. This feature requires that you add an additional URL scheme to your project's settings.

Important: The Native SDK includes a statically linked version of the Facebook SDK. Do not integrate a separate copy of the Facebook SDK into your app.

After you have created your project, open the Info tab for your app's build target. Expand the URL Types section, then click the "+" icon.

Next, change the URL Schemes field to "fb180566588646375<appkey>" where "<appkey>" is the alphanumeric, lowercase-only value at the end of your bundle identifier. For example, if your bundle identifier is "com.mobage.ww.a123.hellomobageios", set the URL Schemes field to "fb180566588646375hellomobageios". The URL type should look similar to the following:



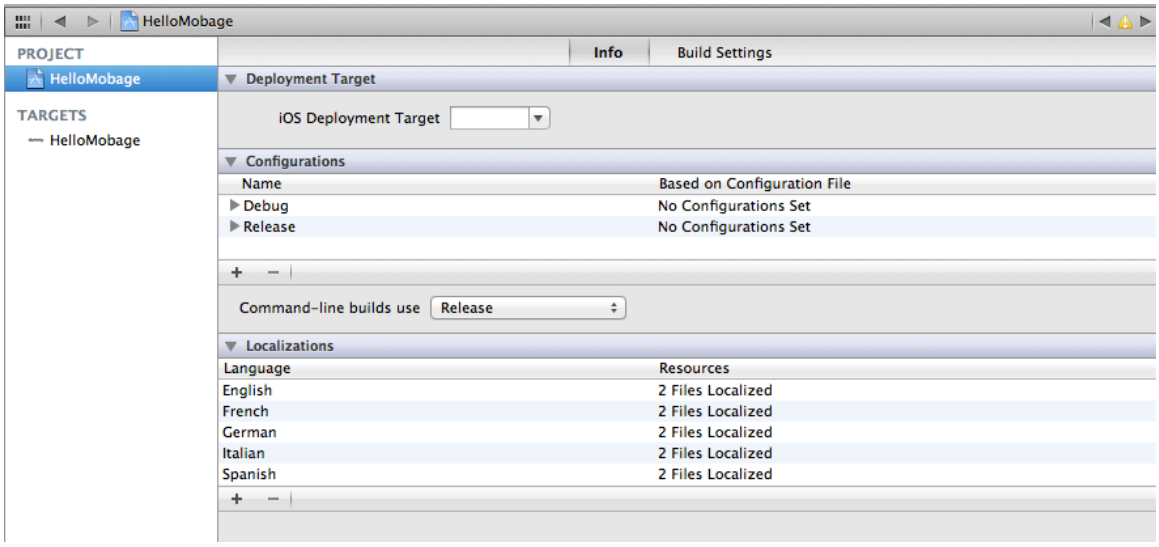
Important: If your bundle identifier changes, you must update the URL type to reflect the new bundle identifier.

ii) Updating localization settings

The Native SDK includes localized text in English, French, German, Italian, and Spanish. To enable the localized text, you must update the project's localization settings:

Click on the Project icon ("HelloMobage") at the top of the left pane, then click the Info tab.

Under Localizations, click the "+" icon, then select "French (fr)." Repeat this step for the "German (de)," "Italian (it)," and "Spanish (es)" localizations. When you are finished, the Info tab should look similar to the following:



NSNotificationCenter considerations

The Native SDK uses the NSNotificationCenter class to dispatch important notifications. Notification names that begin with the string "mobage_" are reserved for the Native SDK. Your app must not generate notifications whose name begins with "mobage_".

For example, your app could create a notification similar to the following:

```
[[NSNotificationCenter defaultCenter]
 addObserver:self
 selector:@selector(notificationReceived:)
 name:@"my_notification_name" object:self];
```

In contrast, the following notification name is not permitted:

```
[[NSNotificationCenter defaultCenter]
 addObserver:self
 selector:@selector(notificationReceived:)
 name:@"mobage_notification_name" object:self];
```


4. Go Social!

In this section, we will look at how to initialize the Mobage framework, how to report your app's lifecycle events to Mobage, and how to establish a user session and call a Social API to make sure that everything is working. Finally, we will look at how to add the Mobage Community Button to your app. For further information on the various API calls that are available, please refer to the API reference.

Initializing the Mobage framework

To initialize the Mobage framework, you must call `[MBMobage initializeMobageWithServerEnvironment:]`. The parameters to this method specify whether to use the sandbox (testing) or production (live) Mobage servers. The parameters also specify the Mobage app key, the app version, and the consumer key/secret pair. You can get the app key and the consumer key/secret pair from the Mobage Developer Portal (https://developer.mobage.com/) after creating an app.

The framework should be initialized only once, when the app starts up. In most cases, you should initialize the framework in the app delegate object's `[UIApplicationDelegate application:didFinishLaunchingWithOptions:]` callback.

In AppDelegate.m (or the equivalent file in your existing project), add the following initialization method call in the `[UIApplicationDelegate application:didFinishLaunchingWithOptions:]` callback. Be sure to replace the `YOUR_*` placeholders with your own values.

Important: Before you submit your app to Mobage for review, you must change the initialization method call to refer to Mobage's production environment.

In AppDelegate.m:

```
#import <MobageNDK/MobageNDK.h>// ...

- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    // ...
    [self.window makeKeyAndVisible];

    // This example initializes Mobage with the sandbox (test) environment.
    [MBMobage initializeMobageWithServerEnvironment:kMBServerEnvironmentSandbox
              /* e.g., My-App-iOS */          appId:@"YOUR_APP_KEY"
                   /* e.g., 1.3.8 */     appVersion:@"YOUR_VERSION"
            /* alphanumeric value */    consumerKey:@"YOUR_CONSUMER_KEY"
            /* alphanumeric value */ consumerSecret:@"YOUR_CONSUMER_SECRET"];

    return YES;
}
```


Establishing a user login session

Once you have initialized Mobage, you can add code to log the user into Mobage.

Important: In general, you cannot call the Mobage APIs until the user has logged in.

The Mobage Native SDK stores the user's login session internally so the user does not need to enter their credentials and log in every time. The API method that opens the login dialog, which is `[MBSocialService executeLoginWithCallbackQueue:onComplete:]`, also checks if there is an existing session. If there is an existing session, the login dialog will not be displayed, and the session will be resumed.

In the example below, we establish the login session immediately after Mobage is initialized, so your players can immediately begin accessing the social graph. However, this is not a requirement, and you are free to place the login wherever it provides the best overall user experience.

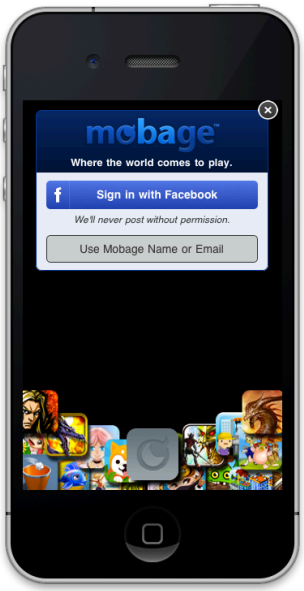
In AppDelegate.m:

```
#import <MobageNDK/MobageNDK.h>
// ...
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    // ...
    [self.window makeKeyAndVisible];

    [MBMobage initializeMobageWithServerEnvironment:kMBServerEnvironmentSandbox
                                              appId:@"YOUR_APP_KEY"
                                         appVersion:@"YOUR_VERSION"
                                        consumerKey:@"YOUR_CONSUMER_KEY"
                                     consumerSecret:@"YOUR_CONSUMER_SECRET"];

[MBSocialService executeLoginWithCallbackQueue:dispatch_get_main_queue()

    onComplete:^(MBCancelableAPIStatus status, NSObject<MBError> *error){
        switch (status){
                case MBCancelableAPIStatusError:
                        NSLog(@"An error occured during user login: %@",error);
                        break;
                case MBCancelableAPIStatusCancel:
                        NSLog(@"User login cancelled.");
                        break;
                case MBCancelableAPIStatusSuccess:
                default:
                        NSLog(@"User login success.");
                        break;
        }
       }];

    return YES;
}
```

Feel free to try running your app at this point and check that everything is working as expected. If this is the first time you run the app, you will be greeted with a login dialog. On the other hand, if you have logged in before, the last session will be recovered automatically.

You can either sign in with a Facebook account or log in with an existing Mobage account. If you don't already have a Mobage user account, you can create one by tapping Use Mobage Name or Email, then Sign Up.

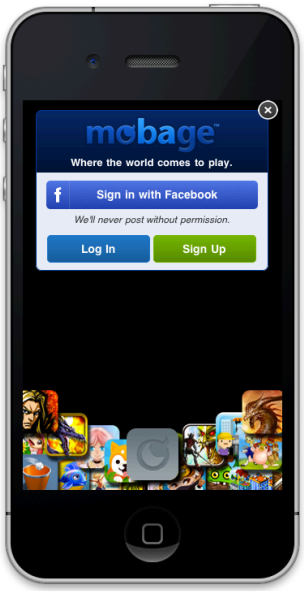
1



Main login dialog.

- Tapping “Sign in with Facebook” shows the Facebook login dialog (not shown).
- Tapping “Use Mobage Name or Email” shows screen (2).

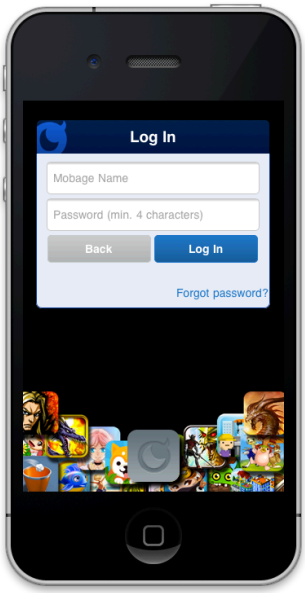
2



After tapping on “Use Mobage Name or Email,” the Mobage “Log In” and “Sign Up” buttons appear.

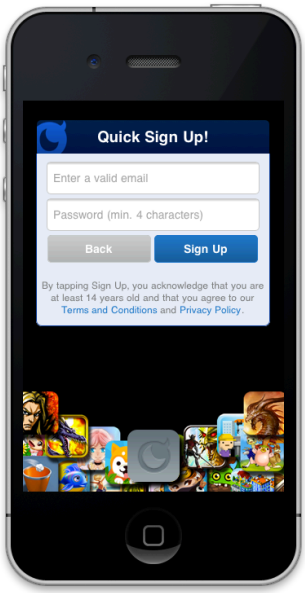
- Tapping “Log In” shows screen (3a).
- Tapping “Sign Up” shows screen (3b).

3a



After entering user credentials and tapping “Log In,” the user can log into Mobage.

3b



After entering an email address and password, then tapping “Sign Up,” the user can sign up for Mobage.

Supporting remote notifications

In version 2.0 and later, the Native SDK enables the use of remote notifications in your app. Remote notifications—also known as push notifications—enable you to send notifications to users in several different ways:

- **Broadcast notifications**: Your app server can use the Mobage REST API to send a broadcast notification to all of the app's users.
- **Server-to-client notifications**: Your app server can use the Mobage REST API to send a single notification to an individual user.
- **Client-to-client notifications**: Your client app can call `[MBRemoteNotification sendToUser:]` to send a single notification to an individual user.

To support remote notifications, your app delegate must include code to handle the following messages:

- `didFinishLaunchingWithOptions:`
- `didRegisterForRemoteNotificationsWithDeviceToken:`

The following example shows how to ensure that your app responds to these messages correctly.

In AppDelegate.m:

```
#import <MobageNDK/MobageNDK.h>

// ...

- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    // ...
 [MBSocialService executeLoginWithCallbackQueue:dispatch_get_main_queue()

    // ...

    [[UIApplication sharedApplication]
        registerForRemoteNotificationTypes:(UIRemoteNotificationTypeBadge |
        UIRemoteNotificationTypeSound | UIRemoteNotificationTypeAlert)];

    return YES;
}

- (void)application:(UIApplication *)app
didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken
{
    [MBMobage sharedInstance].deviceToken = deviceToken;
}
```

Calling the People API

Now that we are able to log in to the Mobage platform, let's call some simple APIs to see if everything is working correctly. For further information on the various API calls that are available, please refer to the API reference.

Here, we will use the `MBPeople` class to get information about the current user. Please remember that you can only make this call after a valid user session is established. In the above example, we would call it inside the `case MBCancelableAPIStatusSuccess:` block.

In AppDelegate.m:

```
[MBSocialService executeLoginWithCallbackQueue:dispatch_get_main_queue()
  onComplete:^(MBCancelableAPIStatus status, NSObject<MBError> *error){
  switch (status){
    case MBCancelableAPIStatusError:
      NSLog(@"Auth flow failed %@",error);
      break;
    case MBCancelableAPIStatusCancel:
      NSLog(@"Auth flow cancelled.");
      break;
    case MBCancelableAPIStatusSuccess:
    default:
      [MBPeople getCurrentUserWithCallbackQueue:dispatch_get_main_queue()
         onComplete:^(MBSimpleAPIStatus status, NSObject<MBError> *error,
                      NSObject<MBUser> *user) {
           if (error){
             NSLog(@"getCurrentUser failed: %@", error);
             return;
           }

           UIAlertView *message =
             [[[UIAlertView alloc]
              initWithTitle:@"Welcome to Mobage"
              message:[NSString stringWithFormat:@"%@, welcome to Mobage!",
                       user.nickname]
              delegate:nil
              cancelButtonTitle:@"OK"
              otherButtonTitles:nil] autorelease];
              [message show];
         }];
      break;
  }
       }];
```

When you run this example, the app will display a welcome message that includes the current user's nickname.

Adding the Mobage Community Button

Starting with version 2.0 of the Mobage Native SDK, all games are required to display the Mobage Community Button in a corner of the screen. Appropriately sized assets are available in the SDK's `MobageResources/` directory.

Note: This requirement replaces the optional branding guidelines for version 1.5.

Your app must follow these rules for displaying the Community Button:

1. At a minimum, display the Community Button on both of the following screens:
    a. A startup screen that is part of the startup flow of an app session for all users.
    b. A primary screen that is used frequently in the app, such as a menu screen.
2. Place the Community Button in any corner of the screen, with no gap between the Community Button and the edges of the screen. The same corner must be used throughout the app.
3. Scale the Community Button image's width and height to 11.875% of the screen's shortest edge. For example, if a device's screen measures 640 pixels (px) by 960 px, scale the Community Button to 76 px by 76 px (11.875% of 640).
4. To represent the Community Button, use the appropriate image from the Native SDK's `MobageAssets/` directory. Follow these guidelines to choose the correct image:
    a. A different image is provided for each corner. Choose the image for the corner of the screen your app is using.
    b. Each image is provided in both "normal" and "down" versions. By default, display the "normal" version.
    c. Each image is provided in multiple sizes. Use the smallest image that can be displayed without upscaling. For example, if the Community Button's required size is 76 px by 76 px, use the 128 px version of the image.
5. When the user is touching the Community Button, display the "down" version of the button image, which simulates a pressed appearance.
6. When the user taps the Community Button, call the method `[MBSocialService showCommunityUI]` to display the Community UI.

And here we come to the end of this start-up guide. We hope that you found it informative. Please continue to explore the Mobage Native SDK by referring to the API reference and various documents on our Developer Portal for more in-depth information.

We hope you have a great time developing games on the Mobage platform!

